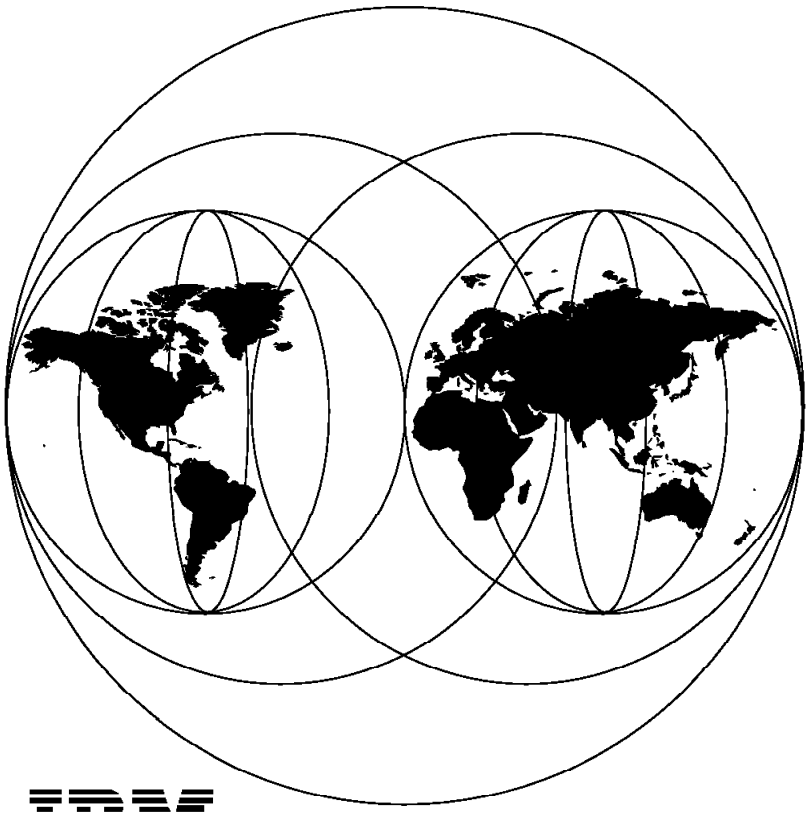
International Technical Support Organization GG24-2501-00

PL/I for OS/2
PL/I for OS/2 Toolkit: Visual PL/I
CODE/370 PL/I Support

March 1995



IBM

International Technical Support Organization
San Jose Center

International Technical Support Organization GG24-2501-00

PL/I for OS/2
PL/I for OS/2 Toolkit: Visual PL/I
CODE/370 PL/I Support

March 1995

Take Note!

Before using this information and the products it supports, be sure to read the general information under “Special Notices” on page xvii.

First Edition (March 1995)

This edition applies to Version 1, Release 1 of both PL/I for OS/2 and PL/I for OS/2 Toolkit, Program Number 5871-AAA, for use with the OS/2 Operating System.

It also applies to Version 1, Release 2 of CoOperative Development Environment/370 (CODE/370), Program Number 5688-194, for use with OS/2 and MVS.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

An ITSO Technical Bulletin Evaluation Form for reader′s feedback appears facing Chapter 1. If the form has been removed, comments may be addressed to:

IBM Corporation, International Technical Support Organization
Dept. 471 Building 070B
5600 Cottle Road
San Jose, California 95193-0001

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Abstract

This document is unique in its detailed coverage of PL/I for OS/2 and Visual PL/I provided with PL/I for OS/2 Toolkit. It focuses on the use of these products for development of MVS host applications or OS/2 applications with a graphical user interface. It provides information about interfacing PL/I for OS/2 with DB2 and CICS and using CODE/370 to develop and test PL/I host applications on the workstation.

This document is written for PL/I application designers and programmers who are moving their development environment from the MVS host to OS/2 workstations. Knowledge of the PL/I language and OS/2 workstations is assumed.

(154 pages)

Contents

Figures

Tables

Special Notices

This publication is intended to help PL/I application designers and programmers use PL/I for OS/2 to develop host, workstation, or client/server applications. The information in this publication is not intended as the specification of any programming interfaces that are provided by PL/I for OS/2 and CoOperative Development Environment/370. See the PUBLICATIONS section of the IBM Programming Announcement for PL/I for OS/2 and CoOperative Development Environment/370 for more information about what publications are considered to be product documentation.

References in this publication to IBM products, programs or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only IBM′s product, program, or service may be used. Any functionally equivalent program that does not infringe any of IBM′s intellectual property rights may be used instead of the IBM product, program or service.

Information in this book was developed in conjunction with use of the equipment specified, and is limited in application to those specific hardware and software products and levels.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594 USA.

The information contained in this document has not been submitted to any formal IBM test and is distributed AS IS. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer′s ability to evaluate and integrate them into the customer′s operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:

ADSTAR
AIX
CICS
CICS OS/2
CUA
DATABASE 2
DB2
DB2/2
DB2/400
DB2/6000
Distributed Database Connection Services/2
IBM
IMS Client-Server/2
MVS/ESA
OpenEdition
Operating System/2
OS/2
Presentation Manager
SQL/DS
System/360
VSE/ESA
Workplace Shell
XGA

The following terms are trademarks of other companies:

Windows is a trademark of Microsoft Corporation.

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Other trademarks are trademarks of their respective companies.

Preface

This document is intended to provide PL/I application designers and programmers with guidelines on the use of PL/I for OS/2 and Visual PL/I provided with the PL/I for OS/2 Toolkit.

It contains a short product description and examples for setting up the workstation environment to work with PL/I programs using relational database access and CICS facilities. It also contains information about the use of CODE/370 to develop and test PL/I host applications on the workstation.

This document is intended for PL/I application designers and programmers who are moving the PL/I development environment from the MVS host to OS/2 workstations.

How This Document Is Organized

The document is organized as follows:

- Chapter 1, “Introduction” provides a description of current and future PL/I development environments and an overview of the products.
- Chapter 2, “Setting Up the Workstation Environment” provides information on how to set up the workstation environment including DB2/2 and CICS OS/2.
- Chapter 3, “Application Scenarios for PL/I for OS/2” provides scenarios for moving two sample PL/I applications from an MVS host to OS/2 workstations.
- Chapter 4, “PL/I for OS/2 Test Facility” describes the graphical testing facility provided by PL/I for OS/2.
- Chapter 5, “Visual PL/I of the PL/I for OS/2 Toolkit” describes in detail Visual PL/I provided with PL/I for OS/2 Toolkit.
- Chapter 6, “WorkFrame/2 and PL/I for OS/2” explains how to use WorkFrame/2 together with PL/I for OS/2.
- Chapter 7, “CODE/370 for Host Testing” explains how to use CODE/370 to develop and test PL/I host applications on the workstation.
- Appendix A, “Sample Code” contains listings of the sample code used in and developed for this redbook.

Related Publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this document.

PL/I for OS/2

- *PL/I for OS/2 Installation*, SX26-3833
- *PL/I for OS/2 Programming Guide*, SC26-8001
- *PL/I for OS/2 Messages and Codes*, SC26-8002
- *PL/I for OS/2 Language Reference*, SC26-8003
- *PL/I for OS/2 Built-In Functions*, SC26-8089
- *PL/I for OS/2 WorkFrame/2 Guide*, SC26-8000
- *PL/I for OS/2 Toolkit Reference*, SC26-8223

Remote File Access

- *DDM Architecture Guide*, SC21-9526
- *ADSTAR Distributed Storage Manager General Information*, GH35-0114

WorkFrame/2

- *IBM C++: IBM WorkFrame/2 Introduction*, S61G-1428

DB2/2

- *IBM DATABASE 2 OS/2 DB/2 Guide*, S62G-3663
- *IBM DATABASE 2 OS/2 Programming Guide*, S62G-3665
- *IBM DATABASE 2 OS/2 SQL Reference*, S62G-3667
- *Formal Register of Extensions and Differences in SQL*, SC26-3316

CICS OS/2

- *IBM CICS OS/2 Installation*, GC33-0879
- *IBM CICS OS/2 Customization*, SC33-0880
- *IBM CICS OS/2 Operation*, SC33-0881

Presentation Manager

- Petzold, Charles. *OS/2 Presentation Manager Programming*. Emeryville: Ziff-Davis Press, 1994.
- *Advanced OS/2 Presentation Manager Programming*, SR28-4646
- *OS/2 Presentation Manager Programming Hints and Tips*, SR28-5598
- *Object-Oriented Interface Design, IBM Common User Access Guidelines*, SC34-4399

CODE/370

- *CODE/370 Installation*, SC09-1624
- *CODE/370 Self-Study Guide*, SC09-2047
- *CODE/370 Debug Tool*, SC09-1623
- *Using CODE/370 with VS COBOL II and OS PL/I*, SC09-1862

International Technical Support Organization Publications

- *SCLM WSP/2 Usage Guide*, GG24-3538
- *Client/Server Computing: The Design and Coding of a Business Application*, GG24-3899
- *MVS/ESA OpenEdition DCE: Application Support Servers, CICS and IMS*, GG24-4482
- *Cooperative Development of High-Level Language Applications Using CODE/370 and LE/370*, GG24-3872

A complete list of International Technical Support Organization publications, with a brief description of each, may be found in:

- *Bibliography of International Technical Support Organization Technical Bulletins,* GG24-3070.

To get a catalog of ITSO technical publications (known as “redbooks”), VNET users may type:

```
TOOLS SENDTO WTSCPOK TOOLS REDBOOKS GET REDBOOKS CATALOG
```

How to Order ITSO Technical Publications

IBM employees in the USA may order ITSO books and CD-ROMs using PUBORDER. Customers in the USA may order by calling 1-800-879-2755 or by faxing 1-800-284-4721. Visa and Master Cards are accepted. Outside the USA, customers should contact their local IBM office.

Customers may order hardcopy ITSO books individually or in customized sets, called GBOFs, which relate to specific functions of interest. IBM employees and customers may also order ITSO books in online format on CD-ROM collections, which contain books on a variety of products.

Acknowledgments

This project was designed and managed by:

- Ueli Wahli, International Technical Support Organization, San Jose Center

The authors of this document are:

- Mark Leung, IBM Australia
- Ueli Wahli, ITSO San Jose Center

Thanks to the following people for the invaluable advice and guidance provided in the production of this document:

- Barry Balk, IBM Santa Teresa Laboratory
- Mike Dedina, IBM Santa Teresa Laboratory

Thanks to Wil Nichols, IBM Raleigh, for the device independent code blocks written in C, which we converted to Visual PL/I.

This publication is the result of a residency conducted at the International Technical Support Organization, San Jose Center.

Chapter 1. Introduction

PL/I started its life in the 1960s as New Programming Language (NPL). However, NPL also stands for the National Physical Laboratory in the United Kingdom. Therefore, NPL was renamed to Programming Language One (PL/I).

Now almost three decades later, 20% of mainframe applications run PL/I (65 billion lines of code), certainly a sign of the reliability of the language.

IBM is evolving the PL/I product family so that you can tap into the benefits of client/server, visual development, and object-oriented technologies, while preserving your investments in PL/I applications and skills. In conjunction with other products, as a PL/I programmer you can access multiple databases on multiple platforms and use tools that encourage code reuse – altogether a cost-effective way of introducing newer and more productive technologies.

PL/I on the Host

PL/I was first implemented on IBM System/360 machines. The most current implementations on VM and MVS are PL/I Version 2 Release 3 and IBM PL/I for MVS and VM (PLI/370).

PL/I for VSE/ESA provides an implementation on the VSE operating system.

PL/I on the Workstation

PL/I on the OS/2 platform was first delivered in 1992 as PL/I PACKAGE/2. It was a simple compiler that provided little integration with other workstation products, such as DB2/2, CICS OS/2, and Presentation Manager (PM).

In July 1994 major enhancements were delivered in the form of PL/I for OS/2 Version 1.1 Professional Edition and PL/I for OS/2 Version 1.1 Personal Edition.

In September 1994 the PL/I for OS/2 Toolkit was delivered. It includes Visual PL/I for PM applications.

See “Product Overview” on page 3 for a description of the features of these new products.

In October 1994 IBM announced PL/I for AIX, which brings an integrated visual development environment to the AIX platform.

PL/I Product Family

The PL/I family of products is comprised of PL/I compilers on host and workstation platforms, and a set of complementary application development, data access, and communications products.

Figure 1 shows the PL/I compilers on various platforms and the products that complement PL/I for OS/2.

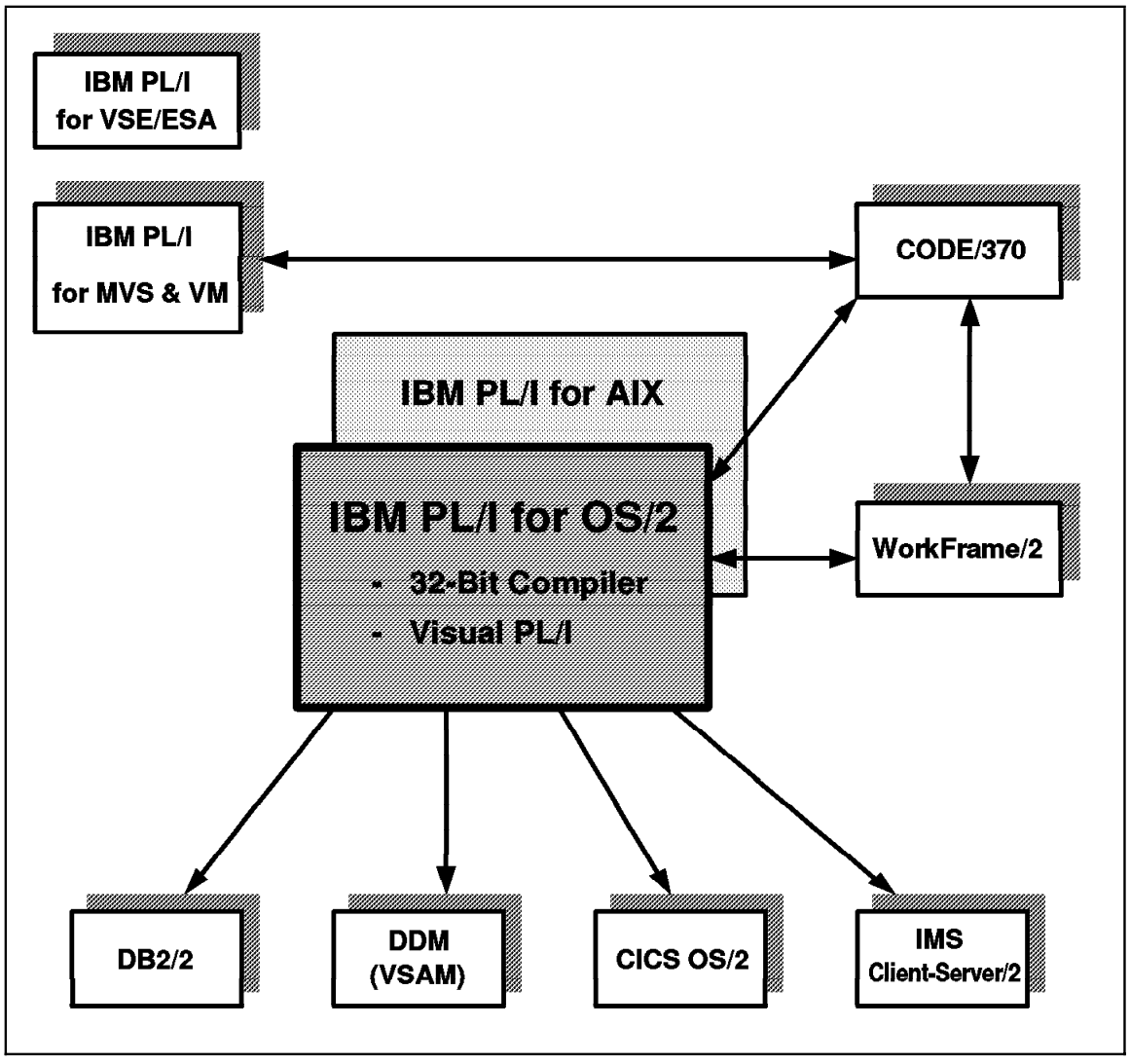


Figure 1. PL/I Family and Complementary Products

IBM PL/I for OS/2 provides a 32-bit compiler and execution environment and Visual PL/I to develop GUI-conforming applications on the OS/2 platform.

PL/I for OS/2 interoperates with WorkFrame/2 to provide an intuitive interactive edit, compile, and debug environment on the workstation. It also interoperates with CODE/370 for edit, compile, and debug of host applications on MVS and VM.

PL/I for OS/2 interfaces with DB2/2 for relational data access, Distributed Data Management (DDM) for local and remote VSAM access, CICS OS/2 for communication and data access on

any CICS supported platform, and IMS Client-Server/2 for access to host IMS data and transactions.

IBM PL/I for MVS and VM is the latest compiler on large mainframes, and IBM PL/I for VSE/ESA provides PL/I on intermediate size mainframes.

The recently announced direction for IBM PL/I for AIX will provide a PL/I compiler and visual development environment on AIX with data and communications access similar to PL/I for OS/2.

Product Overview

In this section we describe the main features and functions of the products used for this redbook, PL/I for OS/2 Version 1 Release 1, PL/I for OS/2 Toolkit, and CoOperative Development Environment/370 (CODE/370) Version 1 Release 2.

PL/I for OS/2 Version 1 Release 1

PL/I for OS/2 Version 1 Release 1 provides a set of new functions over the previous PL/I workstation product:

- A large set of additional language constructs (from the ANSI 87 PL/I standard) to enhance the flexibility and power of PL/I programs.
- PL/I interactive test facility (PLITEST) for animated debugging of your source programs.
- Seamlessly integrated preprocessors for DB2/2, CICS, macro, and include facilities.
- Exception handling enhancements, such as ANYCONDITION, STORAGE, INVALIDOP, and RESIGNAL, to help reduce application downtime.
- PM support, which displays data associated with the keyboard and printer (SYSPRINT) in PM dialog boxes and windows if you link the program as a PM application. Using the PL/I for OS/2 Toolkit you can write full-function PM applications as well.

PL/I for OS/2 Version 1 Release 1 is available in two flavors:

- PL/I for OS/2 Professional Edition
- PL/I for OS/2 Personal Edition.

For this redbook we used PL/I for OS/2 Professional Edition.

PL/I for OS/2 Professional Edition

PL/I for OS/2 Professional Edition provides a full-function OS/2-based PL/I development environment for experienced PL/I programmers. With the Professional Edition you can write client/server applications in PL/I using IMS, CICS, and DB2 subsystems.

Application Development: PL/I for OS/2 Professional Edition facilitates the development of:

- Applications targeted for the workstation or host. This capability is important for customers who are downsizing or offloading host applications. A full development and test environment for host applications with DB2 and CICS is provided on the workstation. MVS, VM, and VSE PL/I applications can be readily ported to OS/2.
- Applications using relational databases, such as DB2. The SQL preprocessor is fully integrated with the PL/I compiler. Using DB2/2 and Distributed Database Connection Services/2 (DDCS/2) you can write client/server applications accessing relational

databases on remote systems. A DCLGEN utility is provided to generate SQL declarations and host variables (data structures).

- Applications using CICS. The CICS preprocessor is fully integrated with the PL/I compiler. Using CICS OS/2 you can write client/server applications enabling transactions on the workstation or on remote host systems. The CICS External Call Interface (ECI) is supported as well.
- Applications using IMS Client-Server/2 with access to IMS transactions on MVS.
- Presentation Manager applications using the PL/I for OS/2 Toolkit.

File Access: PL/I for OS/2 Professional Edition provides access to local and remote files. Standard PL/I file support is available using local OS/2 files. DDM provides a VSAM-like environment for sequential, direct, and keyed access on local or remote systems (through the ADSTAR Distributed Storage Manager, 5648-020).

PL/I for OS/2 Personal Edition

PL/I for OS/2 Personal Edition supports most features of the Professional Edition with the following exceptions:

- SQL preprocessor
- CICS preprocessor
- EBCDIC support
- Support for language level SAA. (Language level SAA causes the compiler to flag keywords and built-in functions not supported by OS PL/I Version 2 Release 3.)

PL/I for OS/2 Personal Edition provides an inexpensive PC or LAN-based application development environment for PL/I. It allows access to PM application programming interfaces (APIs) and C- or REXX-based applications or tools. The PLITEST facility is fully supported in the Personal Edition. The PL/I for OS/2 Toolkit can be used to develop PM applications with a GUI.

PL/I for OS/2 Toolkit with Visual PL/I

The PL/I for OS/2 Toolkit provides an additional set of PL/I-specific tools. It enables you to write real PM applications through Visual PL/I.

You do not have to be an experienced PM programmer with C skills to create GUI applications. You develop your user interface by selecting objects (text, entry fields, push buttons, and so forth) from a palette and then add your application logic using code blocks.

Visual PL/I facilitates reuse of code by providing many predefined reusable code blocks as well as allowing you to create and manage your own code block libraries.

You can define help text in the OS/2 IPF language and use OS/2 color and font palettes to enhance your application.

Visual PL/I manages the development environment for you. It generates the complete PM application, including the PL/I source program, header files, resource files (for PM), help files (for IPF), and a make file to compile and link the program.

PL/I for OS/2 Toolkit provides a utility to convert C header files into PL/I header files. This tool minimizes the effort of incorporating existing C programs into your PL/I applications.

As part of the PL/I for OS/2 Toolkit you also receive the OS/2 Developer′s Toolkit containing the IPF compiler (help facility), PM resource compiler, make facility, import librarian, and enhanced linker.

WorkFrame/2

WorkFrame/2 is delivered as part of PL/I for OS/2. It is an optional part; you can develop PL/I applications without it.

WorkFrame/2 provides an integrated environment for PL/I program development. It is especially useful for programs developed without Visual PL/I, which provides its own development environment.

Using WorkFrame/2 you use intuitive OS/2 workplace shell protocols (drag and drop) to manage PL/I applications on a project-by-project basis. You define appropriate tools and compile and link options for each project. Using WorkFrame/2 you do not have to deal with commands to compile, link, test, and run the PL/I applications.

CODE/370

CODE/370 Version 1 Release 2 provides an edit, compile, and debug environment for host MVS and VM applications written in C, COBOL, or PL/I.

CODE/370 Release 2 provides a language-sensitive editor (LPEX) on the workstation for editing source programs in the supported languages. Programs are automatically downloaded for editing and uploaded for saving in the original host data set.

From the LPEX edit environment you can invoke compile and link operations on the host system and monitor the progress. Error messages are retrieved and matched to the source program in the edit environment for easy correction.

CODE/370 provides the Code Debug Tool, an interactive debugger for the host programs. The mainframe interactive (MFI) version of the Debug Tool provides a fullscreen interface on the host with separate subpanels for source, commands, and monitoring of variable values. The cooperative version of the Debug Tool runs on OS/2 and uses windows to display the source, commands, and variable monitors. The user interface of the debugger is an OS/2 PM application, but the actual program you are testing runs on the host in the target environment, for example, TSO, IMS, or CICS.

As an optional component CODE/370 Release 2 also provides WorkFrame/2. Using WorkFrame/2 you can view your host libraries as OS/2 folders and trigger edit, compile, and link actions through standard OS/2 workplace shell protocols.

We used early driver code of CODE/370 Release 2 for this redbook.

PL/I for OS/2 Future Enhancements

Future enhancements to the PL/I for OS/2 product may include:

- PL/I language-sensitive editor and help facility
- Integration with the next version of WorkFrame/2
- Interlanguage communication with IBM COBOL for OS/2
- Host communications for MVS and VM file access
- Capability to use PL/I objects in IBM VisualAge.

Customer Benefits

Because PL/I will be available on multiple platforms in the very near future, your investment in the language is protected. Using the power of the workstation you will be able to create robust client/server applications with programs and data on multiple platforms.

Investment Protection

When you begin to move to the new workstation environments you can use most of your host PL/I skills to develop your new applications. You can access data (file and relational) on your local machine, as well as connect to other platforms.

Unleash the Power of the Workstation

PL/I for OS/2 provides you with a fast development environment. You can develop and test your local and host applications on the workstation. PL/I for OS/2 provides the PLITEST facility, an interactive debugger with a GUI. You can use Visual PL/I to develop GUI applications on your workstation. Visual PL/I allows you to reuse code in the form of code blocks. Together with WorkFrame/2, PL/I for OS/2 provides an integrated development environment.

Access to Data and Function on Different Platforms

PL/I for OS/2 provides both local and remote file support. Standard PL/I file support is available using local OS/2 files. DDM provides a VSAM-like environment for sequential, direct, and keyed access on local or remote systems.

PL/I for OS/2 provides direct access to DB2/2. You can mirror host databases on DB2/2 or you can use DDCS/2 to access relational data on the host.

PL/I for OS/2 provides direct access to CICS OS/2. Through facilities of CICS OS/2 you can access CICS facilities on other platforms with CICS, such as MVS, VSE, and AIX.

PL/I workstation programs can invoke IMS host transactions using IMS Client-Server/2. This allows for reuse of existing host applications from your workstations using new CUA-compliant GUIs.

IBM DATABASE/2 Software Developer's Kit/2 (SDK/2) allows development of PL/I for OS/2 applications that can later be executed on any OS/2 client running the DATABASE 2 Client Application Enabler/2 product. Such applications can connect to relational databases on other platforms, such as DB2/6000, DB2/400, and SQL/DS.

Investing in the Future with Visual PL/I

Currently most of the books on PM programming are written for C programmers. Visual PL/I comes with working PM code blocks. It is an integrated development environment, generating PL/I code and building executable modules.

Visual PL/I is a learning tool for PM programming, especially for programmers familiar with PL/I on the MVS, VM, and VSE host environments. Visual PL/I is a fun way for PL/I programmers to learn how to exploit the full capabilities of PM programming without having to invest the time to learn a different language.

Chapter 2. Setting Up the Workstation Environment

In this chapter we describe how to prepare the workstation environment to work with the set of PL/I products. We discuss setting up the PL/I for OS/2 Professional Edition and the PL/I for OS/2 Toolkit.

See Chapter 6, “WorkFrame/2 and PL/I for OS/2” on page 79 for instructions on setting up and working with WorkFrame/2.

See Chapter 7, “CODE/370 for Host Testing” on page 91 for instructions on setting up and working with CODE/370.

Installing PL/I for OS/2

PL/I for OS/2 Professional Edition consists of five diskettes. Follow the installation instructions for this product and let the system update your CONFIG.SYS file with the new PL/I directories. We installed PL/I for OS/2 in the following directories:

```
D:\IBMPLI      - PL/I product
D:\IBMDDM      - Record I/O routines
```

Check whether a corrective service diskette (CSD) is available. We installed CSD 2 after we installed the product. CSD 2 adds a PL/I Actions Profile for WorkFrame/2 (see “Updating the Default Actions Profile” on page 80).

Be sure to study the READ.ME file in the IBMPLI directory. It contains some late changes to the product and documentation.

At the end of the installation you will see the IBM PL/I for OS/2 folder on your desktop. See Figure 2.

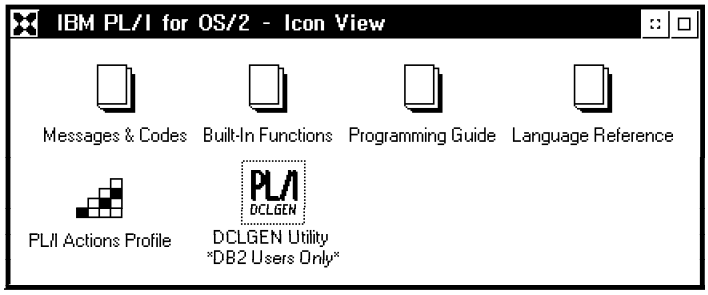


Figure 2. IBM PL/I for OS/2 Folder

In addition there is a set of two diskettes for WorkFrame/2. Installation of WorkFrame/2 is optional; see Chapter 6, “WorkFrame/2 and PL/I for OS/2” on page 79.

Installing the PL/I for OS/2 Toolkit

The PL/I for OS/2 Toolkit consists of two diskettes. In addition there is a set of diskettes for the IBM Developer's Toolkit for OS/2.

If the IBM Developer's Toolkit for OS/2 has not been installed on your workstation, install it.

Follow the installation instructions for the PL/I for OS/2 Toolkit and let the system update your CONFIG.SYS file with the new directories. We installed the PL/I for OS/2 Toolkit in the following directory:

```
D:\PLITK      - PL/I for OS/2 Toolkit
```

We used individual subdirectories of PLITK for each application developed with Visual PL/I.

At the end of the installation you will see the IBM PL/I for OS/2 Toolkit folder on your desktop. See Figure 3.

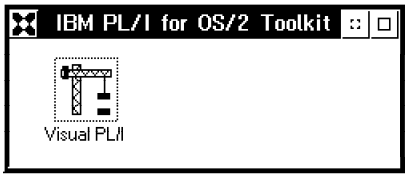


Figure 3. IBM PL/I for OS/2 Toolkit Folder

Setting up DB2/2

If your applications access relational tables, you should install DB2/2. PL/I for OS/2 does not have any special requirements for DB2/2.

For online information on DB2/2 check out the DATABASE 2 OS/2 folder. The DB2 Command Reference icon has most of the information you will need.

Moving DB2 Definitions to the Workstation

If you intend to download PL/I applications with SQL statements from the host (MVS DB2), you need to define and populate appropriate DB2 tables on the workstation or access the host databases using DDCS/2.

For our tests we implemented appropriate DB2 tables on the workstation. In general there is good compatibility between DB2 definitions on MVS and OS/2, but there are differences in the implementation for certain SQL statements. See Chapter 6, "Statements" in *Formal Register of Extensions and Differences in SQL*. For example, the concepts of storage group (STOGROUP) and tablespace are not known in DB2/2.

See Chapter 3, "Application Scenarios for PL/I for OS/2" on page 19 for some of the differences we encountered when implementing DB2 MVS tables on DB2/2.

If you have saved the data definition language (DDL) statements in SPUFI format on the host, download them to the workstation and use the DBM command processor provided by DB2/2. Otherwise you can use the Query Manager to re-create definitions.

DBM Command Processor

The DBM command processor accepts individual DDL or data manipulation language (DML) statements. Before you run any statements you must connect to a database:

```
DBM CONNECT TO database
```

Note: Issue the STARTDBM command if DB2/2 is not active. You are prompted to log on to the workstation if you have not done so already (see Figure 4).

Figure 4. Local Logon

The DBM command processor has a limit of 256 characters as input (imposed by the OS/2 command processor). To submit existing DDL (or DML) statements from a file downloaded from the host we developed a REXX program, RUNSQL. This program reads statements from an input file, removes unnecessary blanks to compress the statement, and submits each statement to the DBM command processor. The program is listed in “Program RUNSQL” on page 115.

The DBM command processor writes an output log to the \SQLLIB\DBM.RUN file.

Sometimes you may try to access a database that another process is accessing. To “free” the database, you can issue a `LOGOFF /L` command. You can then selectively log off the appropriate user IDs as shown in Figure 5.

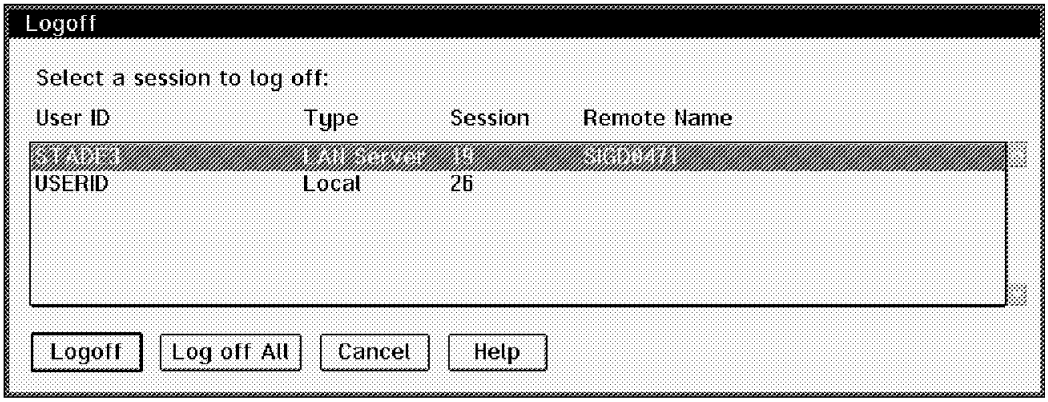


Figure 5. Logoff Utility

Using Views to Mirror the Host

Unlike DB2 on MVS, DB2/2 does not have synonyms. To get around this difference between the two environments, you should create views. For example, the sample database tables may be installed with the prefix USERID (for example, USERID.STAFF), but on MVS the prefix is Q (for example, Q.STAFF). Therefore create a view Q.STAFF for the USERID.STAFF table:

```
DBM CREATE VIEW Q.STAFF AS SELECT * FROM USERID.STAFF
```

Using the DCLGEN Utility

One of the utilities shipped with PL/I for OS/2 is DCLGEN, which creates PL/I include members from existing tables or views.

DB2/2 must be started before double-clicking on the DCLGEN icon in the IBM PL/I for OS/2 folder. If you have not logged on, you will be prompted to log on as well.

First you have to select a database as shown in Figure 6.

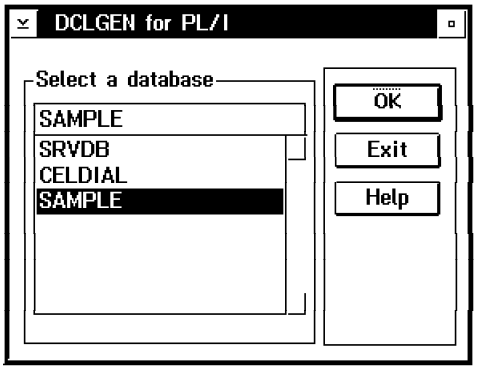


Figure 6. DCLGEN for PL/I: Select Database

Next you enter the table qualifier (Q) as shown in Figure 7.

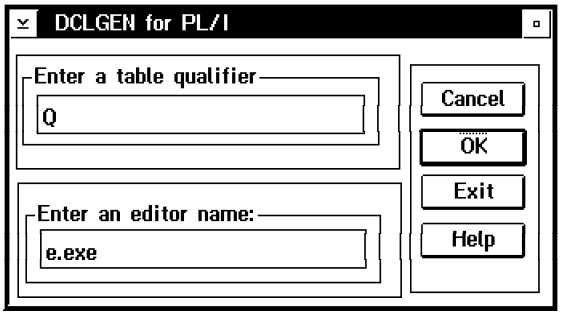


Figure 7. DCLGEN for PL/I: Select Qualifier

DCLGEN displays the list of tables and views. Select a table name and optionally enter the structure name as shown in Figure 8 on page 11.

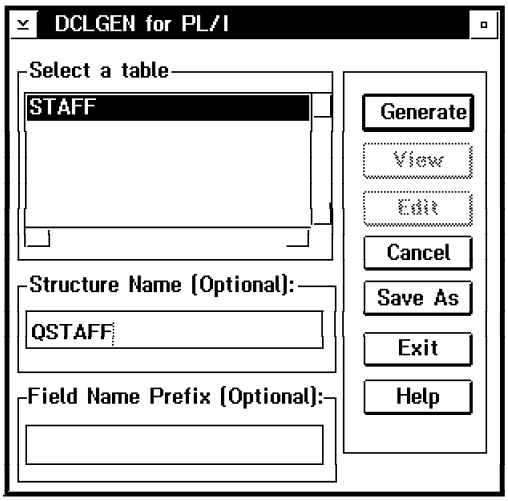


Figure 8. DCLGEN for PL/I: Select Table and Generate

You can now view or edit the generated output and move the file (STAFF.CPY) from \IBMPLI\BIN into an appropriate directory where it can be found using PL/I %INCLUDE statements.

Figure 9 shows the output of the DCLGEN utility for the STAFF table.

```
/************************************************************/
/* DCLGEN COMMAND MADE THE FOLLOWING STATEMENTS             */
/************************************************************/
    EXEC SQL DECLARE Q.STAFF TABLE
    (
     ID                             SMALL NOT NULL ,
     NAME                           VARCHAR(9) ,
     DEPT                           SMALL ,
     JOB                            CHAR(5) ,
     YEARS                          SMALL ,
     SALARY                         DECIMAL(7,2) ,
     COMM                           DECIMAL(7,2)
    ) ;
/************************************************************/
/* PL/I DECLARATION FOR TABLE STAFF                         */
/************************************************************/
DCL 1 QSTAFF,
     5 ID                             FIXED BIN(15),
     5 NAME                           CHAR(9) VAR,
     5 DEPT                           FIXED BIN(15),
     5 JOB                            CHAR(5),
     5 YEARS                          FIXED BIN(15),
     5 SALARY                         FIXED DECIMAL(7,2),
     5 COMM                           FIXED DECIMAL(7,2);
/************************************************************/
/* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 7   */
/************************************************************/
```

Figure 9. DCLGEN for PL/I: Generated Output

Setting up CICS OS/2

To download, test, and run CICS applications on the workstation you need CICS OS/2. You must install CICS 2.0 and at least CSD 3, which includes the PL/I support.

After installation of CSD 3 inspect the CICS environment command, CICSENV.CMD, in the \CICS200\UTIL directory and make sure that it includes the variables for PL/I. You can double-click on the CICSENV.CMD icon in the CICS folder to invoke the editor for the command source.

Figure 10 shows the CICS 2.0 folder.

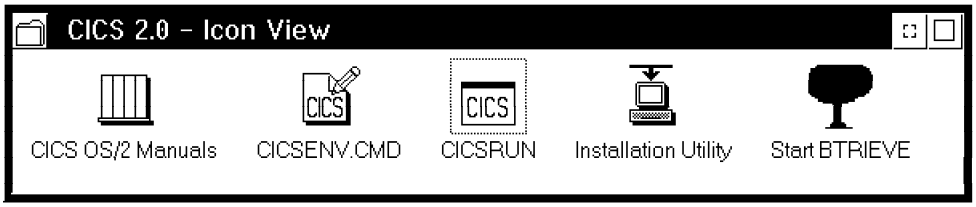


Figure 10. CICS 2.0 Folder

Start CICS by double-clicking on the CICSRUN icon and wait for a CICS terminal window to appear.

Defining Programs and Files

Use the CEDA transaction to define the CICS resources. We suggest that you put your own resources in your own separate resource group.

Here is the abbreviated sequence to define the resources necessary for the sample application used in “CICS/DB2 Application” on page 23:

1. Start CICS, log on as SYSAD, and define a new resource group, for example, CICSPLI. Use CEDA and modify the System Initialization Table (SIT).
2. Shut down CICS using the CQIT transaction.
3. Add the new CICSPLI group to the tables to be loaded by modifying the CICS environment command, CICSENV.CMD, and changing this line:

   ```
   CicsRgrp          =’CICSPLI’
   ```

4. Restart CICS to define the new resources in the CICSPLI resource group, again using CEDA.
5. Change the Program Control Table (PCT) and add the TCLL transaction with PLISTUB as the program to be invoked. A sample definition screen is shown below.

```
  Update    Add       View      Delete                          Exit      Help

FAAPCT2                    Program Control Table-1

       Transaction Code . . . . . . : TCLL

       Group Name . . . . . . . . . : CICSPLI
       Program Name . . . . . . . . : PLISTUB

       Task Class . . . . . . . . . : N              (1-10 or N)

       Secure . . . . . . . . . . . : N              (Y or N)
       Can Be Purged. . . . . . . . : Y              (Y or N)

       Use Alternate Screen Size. . : N              (Y or N)
       Priority . . . . . . . . . . :   0            (0-255)

       System ID. . . . . . . . . . :
       Remote Transaction Code. . . :

       Description. . . . . . . . . : STUB TO CALL OTHER PL/I PROG.


Enter F1=Help F3=Exit               F10=Actions F12=Cancel
```

6. Change the Destination Control Table (DCT) and add the extra-partition data set, PLIO, for test output of the two PL/I programs. A sample definition screen is shown below.

```
  Update    Add       View      Delete                          Exit      Help

FAADCT2         Destination Control Table-1

Description     Destination. . . . : PLIO
                Group Name . . . . : CICSPLI
                Facility Type. . . : E              (L, R, I, E)
                Description. . . . : OUTPUT FROM PL/I PROGRAMS

Intra-partition Recoverable Queue. :                (R or U)
                Trigger level. . . :
                Transaction ID . . :         Terminal ID. . . . :

Extra-Partition Input/Output . . . : O       Fixed/Variable . . : V
                Device type. . . . : F       Max record length. :   132
                Device/file name . : CICSPLI.OUT

Remote          System ID. . . . . :
                Record Length. . . :
                Remote Destination :

Indirect        Indirect Dest. . . :

Enter F1=Help F3=Exit               F10=Actions F12=Cancel
```

7. Shut down CICS to make the new definitions active at the next restart of CICS.

Preparing PL/I Compile Options

Compile options are usually necessary to compile PL/I source programs with appropriate options to specify required output, optimization, and testing.

The *PL/I for OS/2 Programming Guide* contains chapters on setting compile-time options:

- Chapter 4, “Preparing to compile your program,” discusses the different methods of specifying compile-time options.
- Chapter 5, “Compile-time option descriptions,” explains the meaning and usage of each option.
- Chapter 6, “PL/I preprocessors,” describes the INCLUDE, MACRO, SQL, and CICS preprocessors. See “Preprocessors” on page 15 for more details.

Table 1 shows the different methods of specifying compile-time options, using examples relevant to people developing applications using PL/I for OS/2, PL/I for OS/2 Toolkit with Visual PL/I, and WorkFrame/2. Each successive method in the table overrides the previous method.

Table 1. Specifying Compile Options

Method	Example
1 Environment variables IBM.OPTIONS and IBM.PPxxx	Set in CONFIG.SYS file. Applies to all compilations.
	Set from the OS/2 command line. Applies only to compilations run in that session. Options can be set by a command file such as PLIPPENV.CMD (see “Program PLIPPENV - Set PL/I Environment Variables” on page 116).
2 Compile options at invocation	Coded by hand at compiler invocation, for example: PLI MYPROG.PLI (TEST
	Coded in a command file used to invoke the compiler, for example, CICSPSQC.CMD (see “Program CICSPSQC - Compile a PL/I SQL CICS Program” on page 117).
	Options in the project settings in Visual PL/I. These are added to the make file for the project, and the NMAKE utility invokes the PL/I compiler with the options you set.
	Options in the project settings in WorkFrame/2. These are added to the make file for the project, and the NMAKE utility invokes the PL/I compiler with the options you set.
3 **%process** or ***process** statements	Coded by hand at the beginning of the PL/I source.
	Added as code blocks using Visual PL/I. Code blocks are provided by Visual PL/I or created by the designer as My Code Blocks (refer to “My Code Blocks” on page 47).

PL/I Environment Variables

PL/I for OS/2 uses a set of environment variables to direct the PL/I compiler. The full set of variables is described in “Setting compile-time environment variables” in the *PL/I for OS/2 Programming Guide*. The most important variables are listed below.

IBM.OPTIONS Set of compile options to be used for every compilation.

IBM.SOURCE Path used to find the source program to be compiled, if the name is not qualified with a directory.

IBM.SYSLIB Directory search path for include files (%INCLUDE statements). This path is searched before the INCLUDE environment variable.

INCLUDE Standard directory search path of OS/2 for included files; used by PL/I after the IBM.SYSLIB paths.

IBM.PRINT Directory path for listing files; default is the current directory.

IBM.OBJECT Directory path for object files; default is the current directory.

IBM.PPSQL Options for the SQL preprocessor.

IBM.PPCICS Options for the CICS preprocessor.

There are also environment variables for the execution of PL/I programs. See “Specifying run-time options” and “Specifying characteristics using DD.ddname environment variables” in the *PL/I for OS/2 Programming Guide*.

CEE.OPTIONS PL/I run-time options, such as HEAP, INTERRUPT, RPTOPTS, and RPTSTG.

DD.ddname Directory and file name for input and output files used in the program. The ddname is either the file constant or the alternate ddname specified in the TITLE option of the OPEN statement. The value assigned to the environment variable contains the OS/2 file name and additional information, such as BUFSIZE, RECSIZE, SHARE, and TYPE.

```
SET DD.input1 = d:\dir\file.ext,recsize(80),share(read),type(crlf)
```

Preprocessors

PL/I source programs using either SQL or CICS must first be run through the appropriate preprocessors before they can be compiled using the PL/I compiler.

On the host, preprocessors are invoked as separate steps, and the source code generated from a preprocessor is passed to the next preprocessor or compiler.

PL/I for OS/2 provides a much better integration. Preprocessors are invoked by compile options, and the PL/I compiler runs the preprocessors directly.

The preprocessors are:

- The **INCLUDE preprocessor**, which resolves %INCLUDE statements. It is sometimes necessary to run this preprocessor before invoking the SQL or CICS preprocessor if other source code must be included first.
- The **MACRO preprocessor**, which provides a full macro facility that can generate additional source code.
- The **SQL preprocessor**, which takes care of all EXEC SQL statements to access relational data.
- The **CICS preprocessor**, which takes care of all EXEC CICS statements to access CICS facilities.

You invoke preprocessors by using the PP compile option. The general format is:

```
PP ( preprocessorname(options) preprocessorname preprocessorname(options) )
```

For example, to compile a PL/I CICS program with SQL you would code:

```
   PP ( SQL CICS MACRO )
or
   PP ( SQL('DBNAME(sample) BIND') CICS('DECK EDF') MACRO )
```

The options for each preprocessor can be specified in the invocation, or they can be stored in environment variables:

```
SET IBM.PPSQL  = 'DBNAME(sample) BIND'
SET IBM.PPCICS = 'DECK EDF'
```

We developed a simple command, PLIPPENV.CMD, to set environment variables before compiling our programs:

```
plippenv database sql-options ( cicsoptions

result:  SET IBM.PPSQL  = DBNAME(database) ISOLATION(CS) BIND sql-options
         SET IBM.PPCICS = cicsoptions
```

See "Program PLIPPENV - Set PL/I Environment Variables" on page 116.

SQL Preprocessor

To allow your PL/I code to access SQL for DB2/2 you need to invoke the SQL preprocessor to translate the source code at compile time.

As part of the options for the SQL preprocessor you must pass the database name, for example, SAMPLE. We also passed the BIND parameter to create a bind file (extension .bnd), which can be bound to the database using the SQLBIND utility afterwards.

Your PL/I program must connect to the database:

```
EXEC SQL CONNECT TO SAMPLE;
```

This change is necessary when migrating your programs from the host environment.

CICS Preprocessor

To allow your PL/I code to access CICS OS/2 you need to invoke the CICS preprocessor to translate the EXEC CICS statements into the appropriate library calls.

Usually you do not need to pass any options to the CICS preprocessor. You can enable or disable the CICS Execution Diagnostic Facility (EDF) and generate a source output file (DECK).

Note: You must invoke the MACRO preprocessor after the CICS preprocessor and you must use the compile option, SYSTEM(CICS).

CICS programs are usually linked into a dynamic link library (DLL) module. The location for the DLL modules is specified in the CICSENV command, and you must run this command before compiling programs. Select a DLL library for your own modules, for example:

```
UserWrk           ='D:\DLLX'
```

Command Procedures for Compiling and Linking

Command procedures for compiling PL/I programs are available in \IBMPLI\BIN:

```
PLI        - compile a PL/I program
PLILINK    - link    a PL/I program into an executable (EXE)
                     (no SQL or CICS libraries included)
```

Command procedures for compiling PL/I CICS programs are available in \CICS200\UTIL:

```
CICSPCMP   - compile a PL/I CICS program
CICSPLNK   - link    a PL/I CICS program
```

These procedures do not cover programs with SQL and CICS. We developed a few additional procedures for PL/I programs with SQL or with SQL and CICS:

```
CICSPSQC   - compile a PL/I CICS program with SQL for a DLL
PLILINKP   - link    a PL/I program into an executable (EXE)
                     (with SQL and/or CICS ECI)
```

The compile procedure, CICSPSQC, is a modified version of CICSPCMD (see “Program CICSPSQC - Compile a PL/I SQL CICS Program” on page 117 for a listing). It passes proper preprocessor options to the compiler as described in “Preprocessors” on page 15:

```
old:  compopts = '(system(cics) pp(cics("deck print") macro) ...
new:  compopts = '(system(cics) pp(SQL cics("deck print") macro) ...
```

The link procedure, PLILINKP, is a modified version of PLILINK. (see “Program PLILINKP - Link a PL/I Program with SQL/CICS” on page 116 for a listing). It includes the necessary library files of DB2 and CICS:

```
CICS library:  FAAPSTRT + FAAPLID + FAAOTSPL + FAACIC32
SQL  library:  SQL_DYN
```

If you have DB2/2 and CICS OS/2 installed we recommend that you modify the link procedure, CICSPLNK, to include the DB2 library, SQL_DYN:

```
old:  /* Set PL/I and CICS libraries required */
      libs = "ibmlink+ceelink+faapstrt+faaplid+faaotspl"

new:  /* Set PL/I and CICS and DB2 libraries required */
      libs = "ibmlink+ceelink+faapstrt+faaplid+faaotspl+sql_dyn"
```

Alternatively you can leave CICSPLNK unchanged and create a modified procedure with the DB2 library included.

Chapter 3. Application Scenarios for PL/I for OS/2

We used two applications from previous redbooks to experiment with offloading host PL/I applications to the workstation. We tried to mirror the host application on the workstation and improve the user interface at the same time. We used a new directory, d:\PLIDATA, for all programs.

The two applications are:

- An ISPF application accessing a DB2 table
- A CICS application reading and updating DB2 tables.

ISPF/DB2 Application

The simple ISPF/DB2 application used to demonstrate the conversion from MVS to OS/2 is described in the redbook, *SCLM WSP/2 Usage Guide*, GG24-3538.

PL/I Programs

The application load module (SCLS101) consists of two source programs:

SCLS101 Main program with ISPF user interface

SCLS102 Subroutine with DB2 access.

The subroutine can be used almost unchanged, whereas the main program must be rewritten to provide a GUI.

DB2 Database

The program accesses the STAFF table provided by the Query Management Facility (QMF). Sample rows of the table are as follows:

ID	NAME	DEPT	JOB	YEARS	SALARY	COMM
10	SANDERS	20	MGR	7	18357.50	---
20	PERNAL	20	SALES	8	18171.25	612.45
30	MARENGHI	38	MGR	5	17506.75	---
40	O'BRIEN	38	SALES	6	18006.00	846.55
50	HANES	15	MGR	10	20659.80	---
60	QUIGLEY	38	SALES	---	16808.30	650.25

Host Application

The host application is invoked through the following commands:

```
DSN SYSTEM(DB23)                      <=== DB23 is the DB2 system ID
RUN PROGRAM(SCLS101) PLAN(SCLS102)    <=== Plan of DBRM of SCLS102
END                                   <=== End the DSN processor
```

The main program displays an ISPF panel as shown in Figure 11.

```
SCLPS001 ------------------------ SCLM Sample Program ------------------------

           Find data of people in Q.STAFF


   Enter ID of person about whom you wish to get data: ====> 20

           ID  NAME         DEPT  JOB     YEARS     SALARY       COMM









  PF1=Help         PF3=End
```

Figure 11. ISPF/DB2 Application Panel Input

Enter the number of a person in the STAFF table, for example, 00020. The subroutine is invoked and returns the data from the DB2 table as shown in Figure 12.

```
SCLPS001 ------------------------ SCLM Sample Program ------------------------

          Find data of people in Q.STAFF


  Enter ID of person about whom you wish to get data: ====> 00020

          ID  NAME         DEPT  JOB     YEARS     SALARY       COMM

       00020  PERNAL         20  SALES       8   18171.25     612.45


 O.k., this is your data

 PF1=Help         PF3=End
```

Figure 12. ISPF/DB2 Application Panel Result

The host application also provides help text in an additional ISPF panel.

The host application does not use indicator variables. When a table row is retrieved that contains null values you get the error message:

```
Request failed for this ID, SQLCODE = -00305
```

This could obviously be improved by better programming! In addition there are differences between DB2 on MVS and OS/2 with regard to the values that are put in the host variables if such an error occurs.

Moving the ISPF/DB2 Application to the Workstation

The steps involved in moving the ISPF/DB2 application to the workstation are discussed below. The assumption is that DB2/2 is installed on the workstation.

Moving DB2 Definitions

On the workstation DB2/2 provides a similar sample database. The sample database (SAMPLE) is installed by running the SQLSAMPL program that DB2/2 provides. The sample database is installed under the userid that is active when the command is run, usually USERID.

To use the same naming convention as on the host, Q.STAFF, we define a view on the workstation as described in “Using Views to Mirror the Host” on page 10. A view can be defined by using the Query Manager of DB2/2 or the DBM command processor:

```
DBM CONNECT TO SAMPLE
DBM CREATE VIEW Q.STAFF AS SELECT * FROM USERID.STAFF
```

Generate the QSTAFF include member for the table as shown in “Using the DCLGEN Utility” on page 10.

Moving the PL/I Programs

The PL/I programs are downloaded to the workstation. We can reuse the subroutine, SCLS102, with a minimal change; we have to connect to the database before issuing any SQL statements:

```
EXEC SQL CONNECT TO SAMPLE;
```

The SCLS102 program is listed in “Program SCLS102” on page 122.

To test the subroutine we reduce the main program, SCLS101, to a minimum and just call the subroutine with a value provided from the OS/2 window (see Figure 13 on page 22).

```
SCLS101: PROC OPTIONS(MAIN) ;
/*********************************************************************/
/**   Reduced version to invoke SCLS102 on the workstation          **/
/*********************************************************************/
dcl SCLS102         ENTRY EXTERNAL,
    cid             bin fixed(15)      init(0),
    cname           char(9) var        init(' '),
    cdept           bin fixed(15)      init(0),
    cjob            char(5)            init(' '),
    cyears          bin fixed(15)      init(0),
    csalary         fixed decimal(7,2) init(0),
    ccomm           fixed decimal(7,2) init(0),
    csqlcode        bin fixed(31)      init(0);
    do until(cid=0);
       get list(cid);
       if cid ¬= 0 then do;
          call SCLS102 ( cid, cname, cdept, cjob, cyears,
                         csalary, ccomm, csqlcode );
          display ('STAFF-TABLE RESULTS:    SQLCODE=' || csqlcode);
          display ('  ID     : ' || cid);
          display ('  NAME   : ' || cname);
          display ('  DEPT   : ' || cdept);
          display ('  JOB    : ' || cjob);
          display ('  YEARS  : ' || cyears);
          display ('  SALARY : ' || csalary);
          display ('  COMM   : ' || ccomm);
       end;
    end;
END SCLS101 ;
```

Figure 13. Reduced Main Program SCLS101

Compile and Link the Programs

To compile the subroutine SCLS102 we need to invoke the SQL preprocessor. We use the commands:

```
SET IBM.PPSQL = DBNAME(SAMPLE) BIND
PLI SCLS102 ( PP(SQL)
```

Note: The BIND file (scls102.bnd) generated by the DB2 preprocessor can be used to bind the program to the SAMPLE database at any time in the future:

```
SQLBIND SCLS102.BND SAMPLE
```

We compile the reduced version of the main program with:

```
PLI SCLS101
```

We link the programs into an executable, SCLS101.EXE, using the command:

```
PLILINKP SCLS101 SCLS102
```

Note: The standard PLILINK command does not include the DB2 library (sql_dyn).

Test the Application

See Chapter 4, “PL/I for OS/2 Test Facility” on page 31 for information about how to use the PLITEST facility to test the application step by step.

Run the Application

We run the application by entering SCLS101 and get the expected results:

```
[D:\plidata]scls101
:                                                  <=== prompt for input
20                                                 <=== enter id
STAFF-TABLE RESULTS:    SQLCODE=             0
  ID     :        20
  NAME   : Pernal
  DEPT   :        20
  JOB    : Sales
  YEARS  :         8
  SALARY :   18171.25
  COMM   :     612.45
:
0                                                  <=== enter 0 to terminate
```

Replacing the ISPF Front End with Visual PL/I

Now we are ready to replace the ISPF front end with a GUI front end using Visual PL/I from the PL/I for OS/2 Toolkit. This activity is described in “GUI Front End for ISPF/DB2 Application” on page 64.

CICS/DB2 Application

To test the downsizing of a PLI/CICS/DB2 application we chose an example used in the redbooks *Client/Server Computing: The Design and Coding of a Business Application*, GG24-3899, and *MVS/ESA OpenEdition DCE: Application Support Servers CICS and IMS*, GG24-4482.

PL/I Programs

From the large set of examples of this application we extracted two PL/I programs:

PCASH CICS program to retrieve cashier information for a given cashier identification from a DB2 table into the CICS communications area. See “Program PCASH” on page 124 for a listing of the program.

PNEWOR3 CICS program to update order information in several DB2 tables from the CICS communications area. See “Program PNEWOR3” on page 126 for a listing of the program.

In addition we wrote two new PL/I programs:

PLISTUB CICS transaction program to test PCASH and PNEWOR3. The CICS transaction, TCLL, is used to invoke the PLISTUB program. The PLISTUB program is basically a copy of the original COBOL program, CALLSTUB, rewritten in PL/I. See “Program PLISTUB” on page 131 for a listing of the program.

PLIECI A PL/I main program that uses the CICS ECI to invoke the two CICS programs, PCASH and PNEWOR3. See “CICS External Call Interface” on page 29 for a description of the CICS ECI and “Program PLIECI” on page 133 for a listing of the program.

DB2 Database

The DB2 database of the sample application consists of many tables. We extracted the following tables used by the PL/I programs:

CASHIER Cashier information; id, name, branch code.

ORDER Order information; bank, branch, cashier, customer, total value.

ORDER_ITEM Items of the order; currency, exchange rate, value.

ORDER_DENOM Denominations used within each order item; bills, value.

Moving the Application to the Workstation

The steps involved in moving this application to the workstation are discussed below. The assumptions for the workstation are that DB2/2 and CICS OS/2 (with the CSD that provides PL/I support) are already installed.

Moving the DB2 Tables of the CICS/DB2 Application

We defined the database and the selected tables with indexes on the workstation. We used the DBM command processor to run the SQL DDL statements as described in “DBM Command Processor” on page 9.

The DDL for the MVS DB2 system was downloaded and modified to fit for DB2/2.

To move the DB2 tables to OS/2:

1. Define the database and connect to it:

```
DBM CREATE DATABASE SRVDB
                    ON E WITH "Database for PL/I for OS/2 Project"
DBM CONNECT TO SRVDB
```

2. Define the tables.

   Some modifications are necessary to define the tables on DB2/2:

   - Remove the IN tablespace option of the CREATE TABLE statements.
   - The table name, ORDER, cannot be used on DB2/2; it is a reserved word. When we tried to create the ORDER table on DB2/2 we got the message:

     ```
     SQL0199N  The use of the reserved word "ORDER" following "CREATE TABLE SRV."
     is not valid.  Expected tokens may include:  "<ID>". SQLSTATE=37501
     ```

     We resolved this problem by changing the table name to ORDERT.
   - A column cannot be defined as FLOAT(7); DB2/2 supports FLOAT.

   The DDL statements are listed below, with the MVS version on the left, and the DB2/2 version on the right.

```
DB2 MVS                                          DB2/2
-----------------------------------------------  ---------------------------------------
                                                 CONNECT TO SRVDB;

CREATE TABLE SRV.CASHIER                         CREATE TABLE SRV.CASHIER
    (CASHIER_NAME        CHAR(40)  NOT NULL,         (CASHIER_NAME        CHAR(40)  NOT NULL,
     CASHIER_ID          CHAR(08)  NOT NULL,          CASHIER_ID          CHAR(08)  NOT NULL,
     BRANCH_CODE         CHAR(08)  NOT NULL)          BRANCH_CODE         CHAR(08)  NOT NULL)
IN SRVDB.SRVTS03;                                   ;

CREATE TABLE SRV.ORDER                           CREATE TABLE SRV.ORDERT
    (ORDER_NUM           INTEGER   NOT NULL,         (ORDER_NUM           INTEGER   NOT NULL,
     ...                                              ...

CREATE TABLE SRV.ORDER_ITEM                      CREATE TABLE SRV.ORDER_ITEM
    (ORDER_NUM          INTEGER    NOT NULL,         (ORDER_NUM          INTEGER    NOT NULL,
     ORDER_ITEM_NUM      SMALLINT  NOT NULL,          ORDER_ITEM_NUM      SMALLINT  NOT NULL,
     TYPE_OF_CURRENCY    CHAR(03)  NOT NULL,          TYPE_OF_CURRENCY    CHAR(03)  NOT NULL,
     XCHNG_RATE          FLOAT(7)  NOT NULL,          XCHNG_RATE          FLOAT     NOT NULL,
     VALU_NEW_CURR       INTEGER   NOT NULL,          VALU_NEW_CURR       INTEGER   NOT NULL,
     VALU_ORIG_CURR      CHAR(20)  NOT NULL)          VALU_ORIG_CURR      CHAR(20)  NOT NULL)
IN SRVDB.SRVTS09;                                   ;

CREATE TABLE SRV.ORDER_DENOM                     CREATE TABLE SRV.ORDER_DENOM
    (ORDER_NUM           INTEGER   NOT NULL,         (ORDER_NUM           INTEGER   NOT NULL,
     ORDER_ITEM_NUM      SMALLINT  NOT NULL,          ORDER_ITEM_NUM      SMALLINT  NOT NULL,
     DENOMINATION        INTEGER   NOT NULL,          DENOMINATION        INTEGER   NOT NULL,
     NUM_BILLS           SMALLINT  NOT NULL,          NUM_BILLS           SMALLINT  NOT NULL,
     VALU                INTEGER   NOT NULL)          VALU                INTEGER   NOT NULL)
IN SRVDB.SRVTS10;                                   ;
```

3. Define the indexes.

   To define the indexes, it is only necessary to change the table name from ORDER to ORDERT in the DDL statements:

```
CREATE UNIQUE INDEX CASHIER_IDX
  ON SRV.CASHIER (CASHIER_ID);
CREATE UNIQUE INDEX ORDER_IDX
  ON SRV.ORDERT (ORDER_NUM);
CREATE UNIQUE INDEX ORDERITM_IDX
  ON SRV.ORDER_ITEM (ORDER_NUM, ORDER_ITEM_NUM);
CREATE UNIQUE INDEX ORDERDEN_IDX
  ON SRV.ORDER_DENOM (ORDER_NUM, ORDER_ITEM_NUM, DENOMINATION);
```

4. Load test data into the tables.

   A few records are loaded into the tables using SQL INSERT statements. This can also be done using the DBM command processor.

   Two cashiers are loaded into the CASHIER table, and two orders (999 and 222) are loaded into the ORDERT table. Order 999 contains two order items, and order 222 contains three order items. Each order item has one to three denominations.

   Cashiers:

```
INSERT INTO SRV.CASHIER (CASHIER_NAME, CASHIER_ID, BRANCH_CODE)
                 VALUES ('TONI HELVETIA', '1234', 'ZRH-MAIN') ;
INSERT INTO SRV.CASHIER (CASHIER_NAME, CASHIER_ID, BRANCH_CODE)
                 VALUES ('WALTER BALANTER', '2222', 'BASEL') ;
```

   Orders:

```
INSERT INTO SRV.ORDERT
  ( ORDER_NUM, VALU_ORD_NEW_CURR, PROFIT, TOTAL, ORDER_STATUS, LAST_DATE_UPD, LAST_TIME_UPD,
    BANK_CODE, BANK_NAME, BANK_COMMISSION, BRANCH_CODE, BRANCH_NAME, BRANCH_ADDRESS,
    CASHIER_ID, CASHIER_NAME, CUST_ID, CUST_NAME, CUST_ADDRESS )
 VALUES ( 999, '6789', '4567', '11356', 'INI', '8/8/1994  ', '10:30:01',
          'SBC', 'SWISS BANK CORP', 1.234, 'ZRH-MAIN', 'ZURICH MAIN OFFICE', 'PARADEPLATZ, ZURICH',
          '1234', 'TONI HELVETIA', 'DNEWS', 'DAILY NEWS', 'COVERED BRIDGE LUCERNE' ) ;
INSERT INTO SRV.ORDERT
  ( ORDER_NUM, VALU_ORD_NEW_CURR, PROFIT, TOTAL, ORDER_STATUS, LAST_DATE_UPD, LAST_TIME_UPD,
    BANK_CODE, BANK_NAME, BANK_COMMISSION, BRANCH_CODE, BRANCH_NAME, BRANCH_ADDRESS,
    CASHIER_ID, CASHIER_NAME, CUST_ID, CUST_NAME, CUST_ADDRESS )
 VALUES ( 222, '2222', '2222', '4444', 'INI', '11/11/1994  ', '11:11:11',
          'SKA', 'SWISS CREDIT BANK', 2.345, 'BASEL', 'SKA BASEL', 'AT THE RHEIN',
```

```
'2222', 'WALTER BALANTER', 'BOATG', 'RHEIN BOATING', 'ON THE RIVER RHEIN' ) ;
```

Order items:

```
INSERT INTO SRV.ORDER_ITEM
        ( ORDER_NUM, ORDER_ITEM_NUM, TYPE_OF_CURRENCY, XCHNG_RATE, VALU_NEW_CURR, VALU_ORIG_CURR )
 VALUES ( 999, 1, 'SFR', 1.33, 665.00, '500.00' ) ;
INSERT INTO SRV.ORDER_ITEM
        ( ORDER_NUM, ORDER_ITEM_NUM, TYPE_OF_CURRENCY, XCHNG_RATE, VALU_NEW_CURR, VALU_ORIG_CURR )
 VALUES ( 999, 2, 'DM', 1.50, 200, '300.00' ) ;
INSERT INTO SRV.ORDER_ITEM
        ( ORDER_NUM, ORDER_ITEM_NUM, TYPE_OF_CURRENCY, XCHNG_RATE, VALU_NEW_CURR, VALU_ORIG_CURR )
 VALUES ( 222, 1, 'CAN', 1.25, 100, '125.00' ) ;
INSERT INTO SRV.ORDER_ITEM
        ( ORDER_NUM, ORDER_ITEM_NUM, TYPE_OF_CURRENCY, XCHNG_RATE, VALU_NEW_CURR, VALU_ORIG_CURR )
 VALUES ( 222, 2, 'SFR', 1.33, 100, '133.00' ) ;
INSERT INTO SRV.ORDER_ITEM
        ( ORDER_NUM, ORDER_ITEM_NUM, TYPE_OF_CURRENCY, XCHNG_RATE, VALU_NEW_CURR, VALU_ORIG_CURR )
 VALUES ( 222, 3, 'FFR', 4.80, 100, '480.00' ) ;
```

Order item denominations:

```
INSERT INTO SRV.ORDER_DENOM ( ORDER_NUM, ORDER_ITEM_NUM, DENOMINATION, NUM_BILLS, VALU )
  VALUES ( 999, 1, 100, 5, 500 ) ;
INSERT INTO SRV.ORDER_DENOM ( ORDER_NUM, ORDER_ITEM_NUM, DENOMINATION, NUM_BILLS, VALU )
  VALUES ( 999, 1, 50, 3, 150 ) ;
INSERT INTO SRV.ORDER_DENOM ( ORDER_NUM, ORDER_ITEM_NUM, DENOMINATION, NUM_BILLS, VALU )
  VALUES ( 999, 1, 5, 3, 15 ) ;
INSERT INTO SRV.ORDER_DENOM ( ORDER_NUM, ORDER_ITEM_NUM, DENOMINATION, NUM_BILLS, VALU )
  VALUES ( 999, 2, 100, 3, 300 ) ;
INSERT INTO SRV.ORDER_DENOM ( ORDER_NUM, ORDER_ITEM_NUM, DENOMINATION, NUM_BILLS, VALU )
  VALUES ( 222, 1, 100, 1, 100 ) ;
 INSERT INTO SRV.ORDER_DENOM ( ORDER_NUM, ORDER_ITEM_NUM, DENOMINATION, NUM_BILLS, VALU )
  VALUES ( 222, 1, 5, 5, 25 ) ;
INSERT INTO SRV.ORDER_DENOM ( ORDER_NUM, ORDER_ITEM_NUM, DENOMINATION, NUM_BILLS, VALU )
  VALUES ( 222, 2, 100, 1, 100 ) ;
INSERT INTO SRV.ORDER_DENOM ( ORDER_NUM, ORDER_ITEM_NUM, DENOMINATION, NUM_BILLS, VALU )
  VALUES ( 222, 2, 10, 3, 30 ) ;
INSERT INTO SRV.ORDER_DENOM ( ORDER_NUM, ORDER_ITEM_NUM, DENOMINATION, NUM_BILLS, VALU )
  VALUES ( 222, 2, 1, 3, 3 ) ;
INSERT INTO SRV.ORDER_DENOM ( ORDER_NUM, ORDER_ITEM_NUM, DENOMINATION, NUM_BILLS, VALU )
  VALUES ( 222, 3, 100, 4, 400 ) ;
INSERT INTO SRV.ORDER_DENOM ( ORDER_NUM, ORDER_ITEM_NUM, DENOMINATION, NUM_BILLS, VALU )
  VALUES ( 222, 3, 20, 4, 80 ) ;
```

5. Disconnect from the database:

```
DBM CONNECT RESET
```

Moving the PL/I Programs

The PL/I programs are downloaded to the workstation. A few changes are necessary as described under “Modifying the PL/I Programs” on page 28.

Include members used by the PL/I programs must be downloaded as well:

PCOMITMS	SQL and program return codes
PCASHCOM	Cashier data in CICS communications area
PORDCOM	Order data in CICS communications area
PMSGCOM	Message data in CICS communications area.

The include members are listed in “Include Members” on page 135.

Host Structures

We used the DCLGEN utility of PL/I for OS/2 to generate the host structures and table definitions from the DB2/2 catalog:

PCASHIER Host structure from CASHIER table

PORDER Host structure from ORDERT table

PORDITM Host structure from ORDER_ITEM table

PORDDEN Host structure from ORDER_DENOM table.

See “Using the DCLGEN Utility” on page 10 for an example of running the DCLGEN utility and “Include Members” on page 135 for a listing of the code.

Setting up CICS OS/2

Prepare the CICS OS/2 system as described in “Setting up CICS OS/2” on page 12. Use the CEDA transaction to define the CICS resources necessary for this application:

- TCLL transaction to invoke the PLISTUB program
- Extra-partition data set, PLIO, in the DCT for test output.

Preparing Program Compilation

To compile PL/I programs under CICS the CICS preprocessor must be invoked. If the program contains SQL statements, the SQL preprocessor and the CICS preprocessor must be invoked. The proper sequence is:

1. Invoke the PL/I INCLUDE preprocessor for %INCLUDE statements (if the SQL or CICS preprocessor is dependent on included code) or replace %INCLUDE statements with EXEC SQL INCLUDE statements.
2. Invoke the SQL preprocessor.
3. Invoke the CICS preprocessor.
4. Invoke the PL/I MACRO preprocessor (required for CICS).
5. Invoke the PL/I compiler.

Compile Options

The compile options for all of the preprocessors are:

```
SYSTEM(CICS) PP(SQL('dbname(srvdb) isolation(cs) bind') CICS('deck print') MACRO)
```

Options for the preprocessors can be either passed in the compile command as illustrated above or stored in environment variables:

```
SET IBM.PPSQL  = DBNAME(SRVDB) ISOLATION(CS) BIND
SET IBM.PPCICS = DECK PRINT
```

When environment variables are used, the compile options can be abbreviated as:

```
SYSTEM(CICS) PP(SQL CICS MACRO)
```

Command Procedures

CICS OS/2 provides REXX commands to compile and link PL/I CICS programs:

CICSPCMP Compile a PL/I program with EXEC CICS statements.

CICSPLNK Link a PL/I program with EXEC CICS statements.

CICS does not provide REXX commands for PL/I programs with SQL and EXEC CICS statements. As described in “Command Procedures for Compiling and Linking” on page 17 we created an additional compile procedure, CICSPSQC, for a PL/I CICS program with SQL, and we modified CICSPLNK to include the DB2 library, SQL_DYN.

We stored the SQL preprocessor options in IBM.PPSQL using a new REXX command, PLIPPENV.CMD (see “Program PLIPPENV - Set PL/I Environment Variables” on page 116 for a listing):

```
command:  PLIPPENV SRVDB
result:   SET IBM.PPSQL  = DBNAME(SRVDB) ISOLATION(CS) BIND
```

Compile and Link

Programs compiled as CICS transactions must be stored as DLLs. The target DLL directory is defined in the CICSENV.CMD under the definition “UserWrk.” We defined a DLL directory, D:\DLLX, added it to the LIBPATH in CONFIG.SYS, and modified CICSENV as shown:

```
UserWrk           =’D:\DLLX’
```

To compile our programs we invoked:

```
CICSPSQC PCASH    ( TEST
CICSPLNK PCASH

CICSPSQC PNEWOR3  ( TEST
CICSPLNK PNEWOR3

CICSPCMP PLISTUB  ( TEST     <=== no SQL statements
CICSPLNK PLISTUB
```

We used the TEST compile option to invoke the PLITEST facility for debugging. To activate the PLITEST debugger at execution time we modified the CICSENV command to specify the TEST execution option:

```
UserPLICeeOptions =’TEST’
```

Modifying the PL/I Programs

To successfully compile the PL/I programs on the workstation some changes are necessary because a connection between CICS and the DB2/2 database must be established and the table name ORDER is a reserved word in DB2/2.

PCASH Program: We added the statement

```
EXEC SQL CONNECT TO SRVDB;
```

in the initialization routine (INIT_DATA).

PNEWOR3 Program: We added the statement

```
EXEC SQL CONNECT TO SRVDB;
```

in the initialization routine (INIT_DATA), and the statement

```
EXEC SQL COMMIT;
```

just before the CICS synchronization point. It was necessary to add the COMMIT statement when we called the program using the CICS ECI facility as described in "CICS External Call Interface." Without the COMMIT statement the table remained locked and the calling program could not reaccess the same data.

We also changed the order table name from ORDER to ORDERT in the update order (UPD_ORDER) routine.

Testing the PL/I Programs

To test the PL/I programs, after starting CICS OS/2 we invoked the TCLL transaction. This started the PLISTUB program, which in turn called the PCASH and PNEWOR3 programs.

See Chapter 4, "PL/I for OS/2 Test Facility" on page 31 for more information about the test and debug facility.

CICS External Call Interface

The CICS OS/2 ECI facility allows a non-CICS program to invoke a CICS transaction. For PL/I programs, CICS OS/2 provides the necessary definitions in an include member called CICS_ECI in the CICS200\PLIHDR subdirectory.

A sample PL/I program called ECIWAIT is provided in the CICS200\SAMPLES\PLI\SOURCE subdirectory.

PL/I CICS ECI Example

We developed a small ECI sample program, PLIECI, to invoke PCASH and PNEWOR3 from outside of CICS. See "Program PLIECI" on page 133 for a listing of the program.

The program performs the following operations:

1. Initializes the ECI parameter structure
2. Initializes the CICS communications area for PCASH
3. Invokes PCASH using the ECI
4. Displays the result of the CICS communications area
5. Initializes the CICS communications area for PNEWOR3
6. Invokes PNEWOR3 using the ECI
7. Displays the result of CICS communications area.

The setup of the ECI parameter structure involves the coding shown in Figure 14 on page 30.

```
%include cics_eci;                  /* ECI include member      */
dcl
   eci_parms_s     type ECI_PARMS,  /* ECI parameter structure */
   sEciReturnCode  fixed bin(15);   /* ECI return code         */
                                    /* ECI stucture initialize */
unspec(eci_parms_s) = ''b;
eci_parms_s.eci_call_type       = ECI_SYNC;     /* synchronous */
eci_parms_s.eci_userid          = "SYSAD";
eci_parms_s.eci_password        = "SYSAD";
eci_parms_s.eci_commarea        = addr(COMMAREA);
eci_parms_s.eci_commarea_length = stg(COMMAREA);
eci_parms_s.eci_timeout         = 60;
                                    /* ECI program to call     */
eci_parms_s.eci_program_name    = "PCASH";
CASHIER_COM.DB_DATA.C_ID = '1234';
                                    /* ECI call                */
sEciReturnCode = FaaExternalCall (eci_parms_s);
                                    /* Display results         */
display ( " - return code: " || sEciReturnCode);
display ( " - cash id    : " || cashier_com.db_data.c_id);
 ...
display ( " - pgm  text  : " || cashier_com.messages.pgm_text);
```

Figure 14. ECI Parameters

Compile and Link: We compiled the PL/I ECI program using the PLI command provided by the PL/I product:

```
PLI PLIECI ( TEST
```

Note that no SQL or CICS preprocessor is needed. We linked the PL/I ECI program into an EXE file using the modified PLILINK command (PLILINKP):

```
PLILINKP plieci
```

Testing the Program: CICS OS/2 must be started before invoking the PLIECI program; otherwise the ECI call fails with a nonzero return code.

Note: The timeout value (eci_timeout) of 60 seconds in the ECI parameter structure may not be enough if the PL/I programs under CICS invoke the PLITEST facility. Increase the timeout value to several minutes when using the PLITEST facility for the called programs.

Adding a Graphical User Interface Using Visual PL/I

The last step of moving the application to the workstation is to develop a GUI using Visual PL/I of the PL/I for OS/2 Toolkit. This step is described in “GUI Front End for CICS/DB2 Application” on page 69.

Chapter 4. PL/I for OS/2 Test Facility

PL/I for OS/2 includes the PLITEST facility for testing PL/I programs. PLITEST uses a GUI to display test information in multiple windows.

In this chapter we describe how to invoke the test facility and provide a few snapshots of the execution of our sample programs. We do not, however, describe all the features of PLITEST. For more information consult the *PL/I for OS/2 Programming Guide*, SC26-8001.

PLITEST can be used for PL/I programs with or without SQL statements running in an OS/2 window environment as well as for programs under CICS.

Note: In the current release PLITEST is not available for PM programs. This includes programs generated with Visual PL/I of the PL/I for OS/2 Toolkit.

TEST Compile and Run-Time Options

The PLITEST facility is controlled by both the TEST compile and TEST run-time options.

Compiling Programs for PLITEST

Programs must be compiled with the TEST compile option for PLITEST to be invoked at execution time. Although you can control the number of hooks put into your object code, it is best to generate the maximum number by specifying TEST, which is equivalent to TEST(ALL,SYM):

```
PLI source.pli ( TEST
```

Executing Programs with PLITEST

You control the execution of PL/I programs with or without the PLITEST facility through the TEST run-time option.

Run-time options are specified in the CEE.OPTIONS environment variable. This variable may be set in CONFIG.SYS or in the session where the PL/I programs are executing:

```
SET CEE.OPTIONS=run-time options
```

Run-time options may also be specified for individual programs through the %PROCESS statement at the beginning of the program:

```
%process runops('run-time options');
```

In this document we concentrate on the TEST run-time option only. See the online *PL/I for OS/2 Programming Guide* for details on all run-time options.

To start the PLITEST facility at the beginning of program execution specify:

```
SET CEE.OPTIONS=TEST                  - defaults
SET CEE.OPTIONS=TEST(ALL,*,:,*)       - suboptions
```

Note: For PL/I programs under CICS the CEE.OPTIONS environment variable is amended by values specified in the CICS environment command, CICSENV.CMD:

```
UserPLICeeOptions ='TEST'
```

Using the PLITEST Facility for Our Sample Applications

We compiled all of our samples with the TEST compile option. Then we invoked the programs using the TEST run-time option.

PLITEST Facility Windows

When a program is invoked with the TEST run-time option, PLITEST displays three windows:

- The OS/2 PLITEST window displays the source code. This window is used to control the execution of the program, set breakpoints, and select variables to display their values.
- The Log window displays variables, data areas, and storage content according to commands entered.
- The Monitor window continuously displays values of selected variables.

OS/2 PLITEST Window

The OS/2 PLITEST window (Figure 15 on page 33) is the main controlling window of the PLITEST facility. It displays the original source code, that is, before any preprocessors have been run.

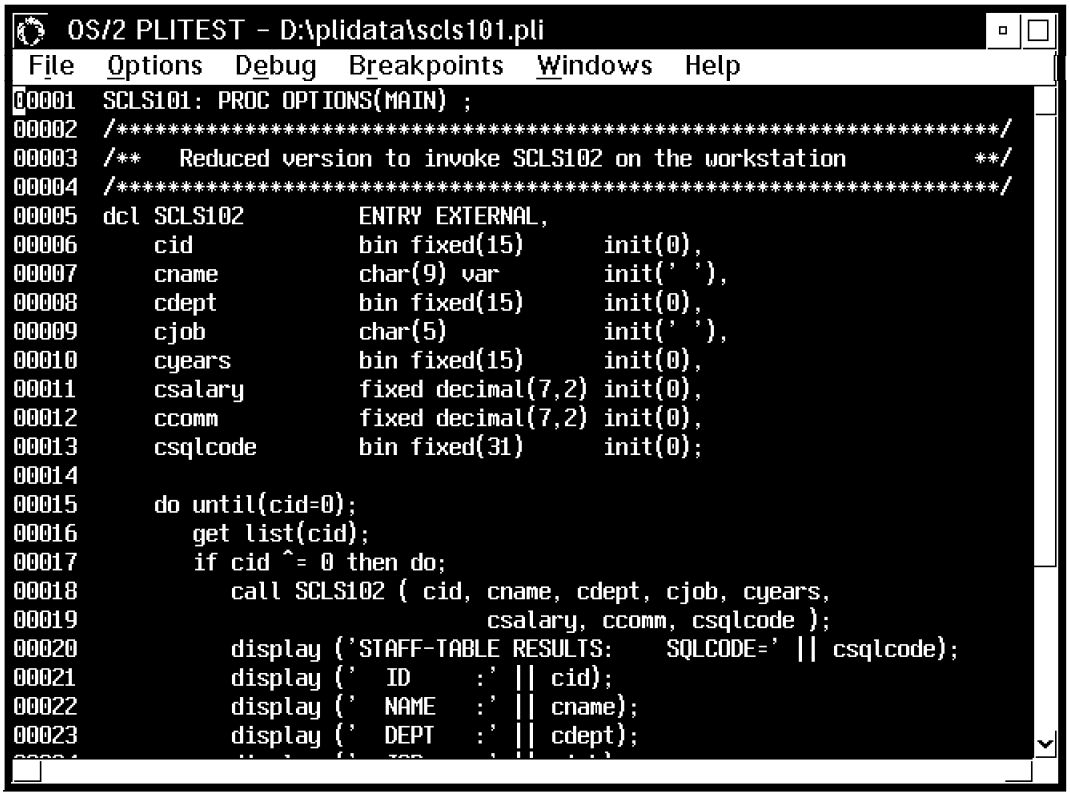


Figure 15. OS/2 PLITEST Main Window

You can use this window to:

- Step through the program (type S).
- Set a breakpoint by double-clicking on a line, or use the Breakpoints pull-down for advanced breakpoints.
- Run to the next breakpoint (type G for “go”).
- List the value of a variable through the Debug pull-down. The value is displayed in the Log window.
- Add a variable to the Monitor window through the Windows pull-down.
- Set the animation rate through the Options pull-down.
- Animate the program at the preset speed (F4-start, F5-stop).
- Toggle to Log and Monitor windows (type T).

At a call to a subroutine the source listing is switched to the new program if that program was compiled with TEST as well.

Log Window

The Log window (Figure 16 on page 34) displays attributes (declaration) of variables, values of variables or expressions (for example, salary + commission), and storage content at an address or pointer variable. It can be used to assign values to variables as well.

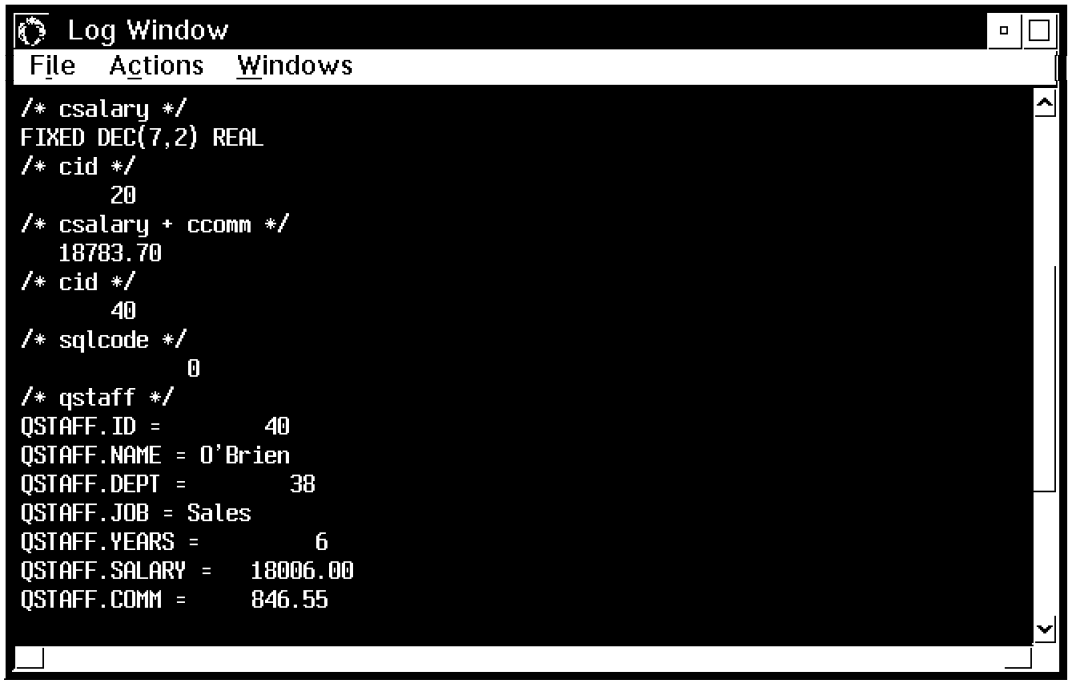


Figure 16. PLITEST Log Window

When PLITEST ends you can save the log content to a file.

Monitor Window

The Monitor window (Figure 17) continuously displays the values of variables that have been added to the Monitor window from the main window or from the Monitor window itself. The format of the display is the same as in the Log window.

You can watch the Monitor window while you step through the program.

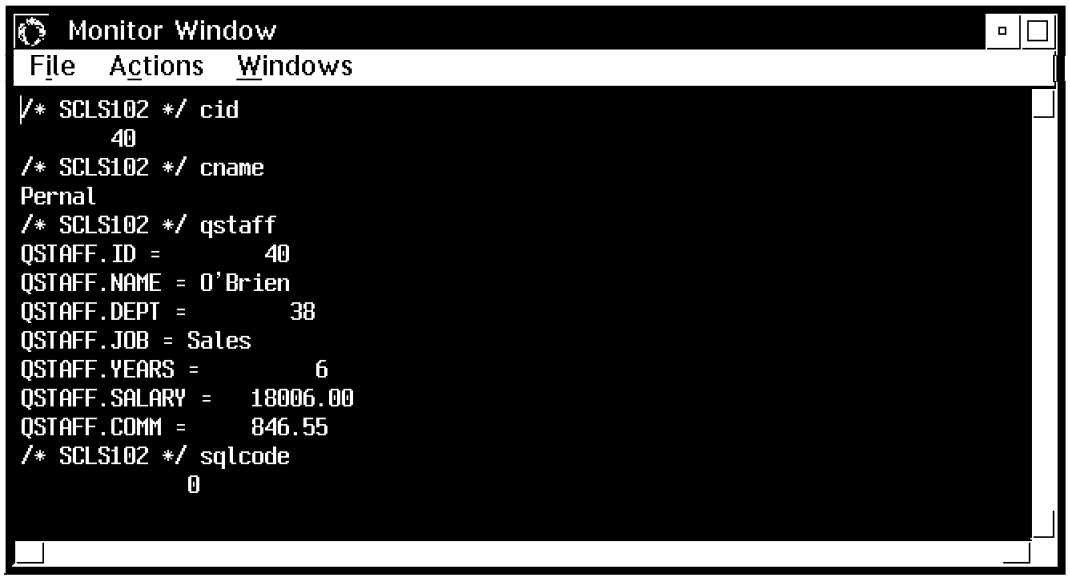


Figure 17. PLITEST Monitor Window

Chapter 5. Visual PL/I of the PL/I for OS/2 Toolkit

In this chapter we look at Visual PL/I, the visual builder provided with the PL/I for OS/2 Toolkit.

Visual PL/I reduces much of the effort required to code PM applications. Therefore application programmers can concentrate on the functionality and usability of their product. However, Visual PL/I does not completely shield the application programmer from having to understand how to write PM programs.

There are many books on PM programming, most of them written for C programmers. Visual PL/I generates working PM code, which can be used for learning PM programming.

Basic Concepts

Visual PL/I code is stored in the d:\PLITK directory and its subdirectories. We recommend that you create a separate subdirectory for each Visual PL/I application because each application consists of about 16 files, and one level of backup files provided by Visual PL/I.

A typical application consists of a master file of the visual definitions (applname.pmg) and a number of generated files:

File	Explanation	Created by
applname.pmg	- visual definitions master	created by Visual PL/I when saving definitions
applname.pli	- PL/I source	generated by Visual PL/I (build and write code)
applname.h	- header file with #defines	″
applname.cpy	- PL/I constants file	″
applname.rc	- PM resource code file	″
applname.def	- PM module definition file	″
applname.ipf	- ipf help file	″
applname.dlg	- dialog file	″
applname.mak	- make file	″
applname.err	- messages of running .mak	generated from running make file (compile)
applname.obj	- object code	″
applname.lst	- compiler listing	″
applname.res	- PM resource file	″
applname.hlp	- help file	″
applname.map	- linkage editor map	″
applname.exe	- resulting executable (or DLL)	″
applname.pl	- own code blocks	created by designer (.pl .plf .plm .plo) (see Figure 31 on page 50)
applname.$xx	- backup files	old files renamed

Starting Visual PL/I

Start Visual PL/I from the IBM PL/I for OS/2 Toolkit folder (see Figure 3 on page 8).

Visual PL/I is loaded into memory and displays the Visual PL/I for OS/2 folder as shown in Figure 18.

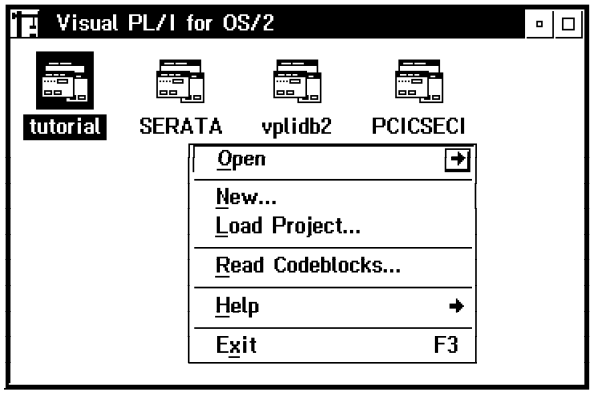


Figure 18. Visual PL/I for OS/2 Folder with Pop-up Menu

Each icon in this folder represents a project. After installation only the tutorial project is in the folder.

Projects can be added to the folder by using the pop-up menu (right mouse button) shown in Figure 18. You can display the Visual PL/I Settings notebook with directory information and compile and link options by selecting the Open action.

To open a project double-click on the appropriate icon. To delete a project use the pop-up menu with the mouse positioned on the project icon.

Pop-Up Menus

Use the right mouse button to display pop-up menus in Visual PL/I. This operation is sensitive to the position of the mouse:

- If the mouse is positioned on the background of a window, the pop-up menu for the entire window is displayed, for example, the pop-up menu of the Visual PL/I for OS/2 folder (Figure 18).
- If the mouse is positioned on an icon within the folder, the pop-up menu for that icon is displayed, for example, the pop-up menu of a project.

This technique is used in all of the windows of Visual PL/I.

Creating a New Project

Before you create your own projects we suggest that you read Part 4, “Tutorials” in the *PL/I for OS/2 Toolkit Reference*.

Use the pop-up menu (Figure 18) to create a new project, and enter a name, for example, STEP1. This creates a new icon in the Visual PL/I for OS/2 folder and automatically opens an empty window with the name of the project and extension, .PMG (STEP1.PMG).

Use the pop-up menu of the new window and select Open-Settings to display the Visual PL/I Project Settings notebook (see Figure 19 on page 37).

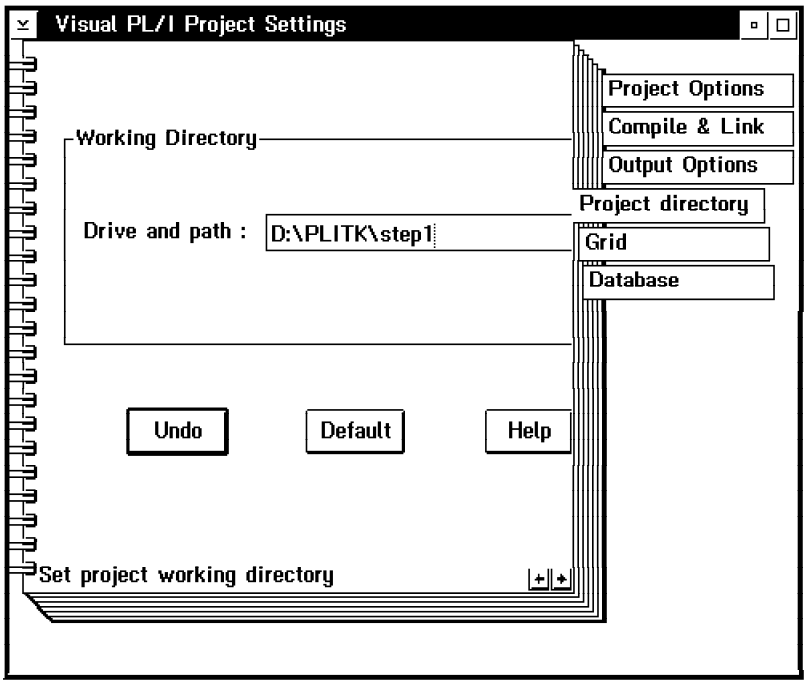


Figure 19. Visual PL/I Project Settings

At this time just set the project directory. We recommend that you assign a separate subdirectory for each project, for example, d:\PLITK\STEP1. The directory is created if it does not exist.

A second small window may also appear on your desktop, the Objects window (see Figure 20). If it is hidden, use the task list (Ctrl-Esc) or Ctrl-O from an open Visual PL/I application window to locate it.

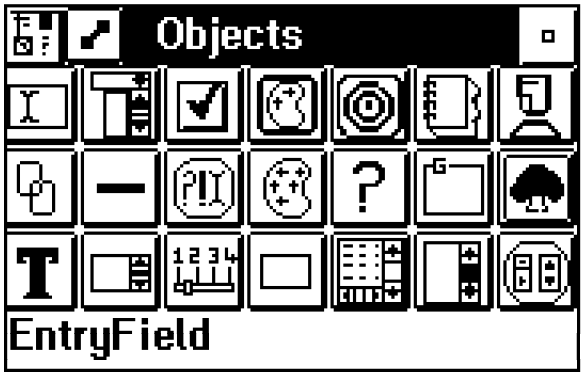


Figure 20. Visual PL/I Objects (Magnified)

The Objects window contains the palette of objects you can drag to windows you are going to design with Visual PL/I. The objects represent entry fields, lists, text, push buttons, notebooks, and so forth.

Move the mouse over the objects (without clicking), and the type of object is displayed at the bottom. Some of the objects can be tailored; see “Customizing Visual PL/I Objects” on page 53.

Designing a Window

Now you can add a main window to the sample project. Use the pop-up menu of the empty project window, select New-Window, and fill in the dialog with the window name, title bar text, and ID (see Figure 21).

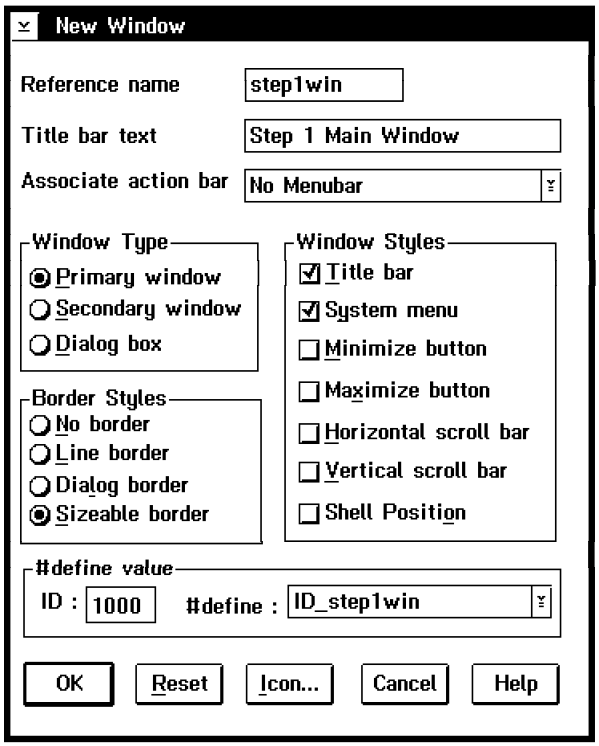


Figure 21. Visual PL/I New Window

An icon for the window appears in the project window (step1win). Save the project from the pop-up menu of the project window (see Figure 22).

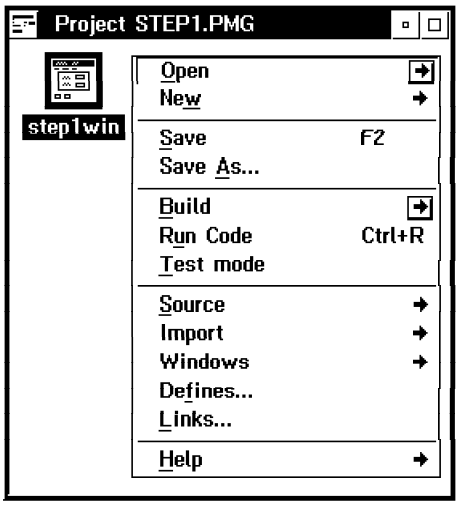


Figure 22. Visual PL/I Project Pop-Up Menu

Open the new window by double-clicking on its icon. You get an empty window where you can start adding your text, fields, and controls.

For this sample we add one text field, one entry field, and one push button. Drag the appropriate objects from the Objects window to the empty window. Figure 23 on page 39 shows a possible layout.

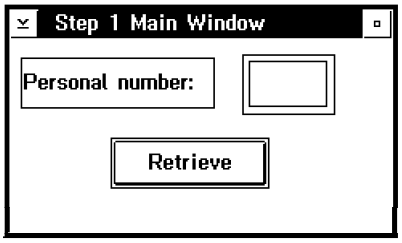


Figure 23. Visual PL/I New Window Layout

Some hints for all new objects:

- Every control has a #define name. Enter a recognizable name so the generated source code will be easy to read.
- You can open the definition of a control again by selecting Change in the pop-up menu of that control.
- Change the appearance by selecting Styles in the pop-up menu. For example, you can make entry fields read-only for output.

Generating the Application

Although we have not added any code yet, we can generate the application to study the layout of the generated program. Always save the project before building.

Use the Build function in the pop-up menu of the project (see Figure 22 on page 38).

Source code is generated and the make file is run automatically. The source is compiled and linked. The progress is visible in an OS/2 window. Figure 24 shows the window with build completed.

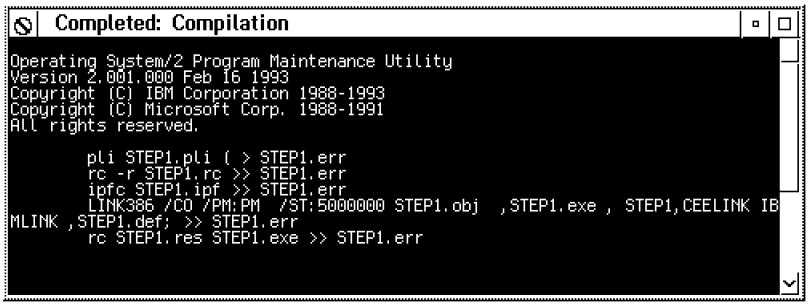


Figure 24. Visual PL/I Project Build Completed

You can run the sample program by typing STEP1 from an OS/2 window in the d:\PLITK\STEP1 directory, or by selecting Run Code in the pop-up menu of the project.

Anatomy of a Visual PL/I Program

You can view the generated code by selecting Source from the pop-up menu of the project. Figure 25 shows the annotated source; blank lines have been eliminated to shorten the listing.

```
Explanation              | Generated Source Program
-------------------------+-----------------------------------------------------------------------------------
Compile options          | *PROCESS MARGINS(1,100) LANGLVL(SAA2) MACRO;
                         | *PROCESS LIMITS(EXTNAME(31));
                         | *PROCESS NOT(ê¬) DFT(BYVALUE);
                         | *PROCESS INCLUDE (EXT('CPY','INC','PLP','MRP'));
                         |  /* ** begin # defines                                              ** */
Standard definitions     | A_name: package ;
                         | %include Pmgdef;
                         | dcl addr        builtin;
                         | dcl binaryvalue builtin;
                         | dcl char     builtin;
                         | dcl iand     builtin;
                         | dcl index    builtin;
                         | dcl ior      builtin;
                         | dcl length   builtin;
                         | dcl null     builtin;
                         | dcl ptrvalue builtin;
                         | dcl stg      builtin;
Standard includes        | %INCLUDE OS2PLI;
                         | %INCL_WIN='Y';
                         | %INCL_DOS='Y';
                         | %INCL_GPI='Y';
                         |  /* ** begin # includes                                             ** */
                         | %INCLUDE OS2;                      /* System Include File      */
                         | %INCLUDE STEP1;                  /* Application Include File */
                         | dcl PMGDisplayDirectory entry(HWND, SHORT, char(*) varyingz byaddr)
                         |           options(byvalue);
                         | %INCLUDE PMGUSER;                   /* VPLI    Include File     */
                         |  /* ** begin prototypes                                           ** */
Window procedure         | dcl STEP1WinProc1 entry(HWND,ULONG,MPARAM,MPARAM)
for my window            |           EXPENTRY returns(optional byvalue MRESULT);
                         |  /* ** begin help prototype/2                                     ** */
                         | /* HWND InitializeIPF (HWND,SHORT,CHAR * , CHAR *,VOID *); */
                         | dcl InitializeIPF entry(HWND byvalue,SHORT,char(*) varz byaddr,
                         |            char(*) varz byaddr,ptr optional) ext returns(HWND byvalue);
                         |  /* ** begin global handles                                      ** */
                         | /* Global Variables */
                         |  /* ** begin global variables                                    ** */
                         |  /* ** begin global entryfield variable declaration             ** */
                         | dcl flCreate1   ULONG;
                         | dcl hwnd1   HWND;
                         | dcl hwndc1 HWND;
                         | dcl hModSTEP1 HMODULE; /* HMODULE hModSTEP1; */
                         | dcl habO HAB;
My entryfield variable   | dcl visid char(5) varyingz;
                         |  /* ** begin help global variable/2                            ** */
                         | dcl hwndOHelpInstance HWND; /* HWND hwndHelpInstance; */
                         |
MAIN PROCEDURE           |  /* ** begin main                                               ** */
-------------------------| UserMain: Procedure options(main);
                         |      /* ** begin local variables                                    ** */
                         |     dcl qmsgO QMSG; /* Message Queue               */
                         |     dcl hmqO HMQ;   /* Handle to Queue             */
                         |     dcl hpsO HPS;   /* Handle to Presentation Space */
                         |      /* ** begin initialize                                       ** */
                         |     /* Intialize PM and obtain anchor block */
 - initialize            |     habO = WinInitialize(0);
                         |      /* ** begin create msg queue                                ** */
                         |     /* Create queue to receive messsages */
 - create message queue  |     hmqO = WinCreateMsgQueue(habO,0);
                         |      /* ** begin register class                                  ** */
                         |     /* Register the window class "STEP1PmgActualWindow1" with procedure STEP1WinProc1*/
                         |     call WinRegisterClass(habO, "STEP1PmgActualWindow1", STEP1WinProc1,
```

Figure 25 (Part 1 of 3). Visual PL/I Generated Code

```
                          |                              CS_SIZEREDRAW, 0);
                          |      /* ** begin set flcreate                                          ** */
                          |     flCreate1=ior( FCF_MINBUTTON ,  FCF_SIZEBORDER ,  FCF_SYSMENU ,  FCF_TITLEBAR);
                          |      /* ** begin create std window                                     ** */
                          |     /* Create a standard window */
 - register my window     |     hwnd1 = WinCreateStdWindow(HWND_DESKTOP, ior( WS_VISIBLE), flCreate1,
                          |                                "STEP1PmgActualWindow1", "Step 1 Main Window", 0,
                          |                                hModSTEP1, ID_STEP1WIN, addr( hwndc1 ));
                          |      /* ** begin set window pos                                       ** */
                          |     /* Set the window position */
                          |     call WinSetWindowPos(hwnd1, HWND_BOTTOM, 12, 163, 246, 147,
                          |                           ior(SWP_MOVE , SWP_SIZE , SWP_ACTIVATE));
                          |      /* ** begin message loop                                        ** */
 * message loop           |     /* Message Loop */
 |                        |     do while ( WinGetMsg(hab0, addr(qmsg0), null(), 0, 0) ¬=0);
 |                        |         call WinDispatchMsg(hab0, addr(qmsg0) );
 |                        |     end;  /* of do */
                          |      /* ** begin destroy window                                       ** */
 - destroy                |     /* Destroy the window */
   everything             |     call WinDestroyWindow(hwnd1);
                          |      /* ** begin destroy ipf/2                                       ** */
                          |     /* Destroy the Help instance */
                          |     /* if (hwnd0HelpInstance) then
                          |        call WinDestroyHelpInstance (hwnd0HelpInstance); */
                          |      /* ** begin destroy msg queue                                   ** */
                          |     /* Destroy the message queue */
                          |     call WinDestroyMsgQueue( hmq0 );
                          |      /* ** begin terminate1                                          ** */
                          |     /* Terminate and release resources */
 - end                    |     call WinTerminate( hab0 );
                          |  /* ** begin terminate2                                          ** */
                          | end;  /* of program */
                          |
 PROCEDURE FOR MY WINDOW  |  /* ** begin prototypes2                                         ** */
 ------------------------ | STEP1WinProc1: proc(hwnd0, msg0, mp1, mp2)
                          |      options( byvalue linkage( system )) returns( MRESULT );
                          |   dcl hwnd0   HWND;
                          |   dcl msg0    ULONG;
                          |   dcl mp1     MPARAM;
                          |   dcl mp2     MPARAM;
                          | /* start of procedure */
                          |     dcl szBased char(1) varz based;
                          |     dcl hps0 HPS; /* HPS hps0; */             /* Handle to Presentation Space */
                          |     dcl rectl0 RECTL; /* RECTL rectl; */         /* Dimensions of window         */
                          |     dcl clr        ULONG;
                          |     dcl attrfound  ULONG;
                          |     dcl ColorTable LONG;
                          |     dcl p          POINTL;
 - message selection      |     select(msg0);
                          |         when (WM_CONTROL)
 * events on my controls  |         do;
 |                        |                 select (SHORT1FROMMP(mp1));
 |                        |                  /* ** begin control case                          ** */
 |                        | when (EF_ID)
 | place for events       | do;
 | in my entry field      |
 |                        |                    end; /* of when  */
 |                        |                  /* ** begin control case                          ** */
 |                        | when (PB_RETRIEVE)
 | place for events       | do;
 | for my push button     |
 |                        |                    end; /* of when  */
 |                        |                otherwise;
 |                        |                end; /* of 2nd level select -2 */
                          |                end; /* of when(WM_CONTROL) */
                          |            when (WM_CREATE)
 * window creation        |            do;
 |                        |                   /* ** begin win create window                   ** */
 |                        |                  /* Create the WC_STATIC, "Personal number:"  */
 | my text constant       |                  call WinCreateWindow(hwnd0, WC_STATIC->szBased, "Personal number:",
 |                        |                                       ior(SS_TEXT ,  WS_VISIBLE ,  DT_VCENTER), 8, 78,
 |                        |                                       119, 30, hwnd0, HWND_TOP, STX_ID, null(), null());
```

Figure 25 (Part 2 of 3). Visual PL/I Generated Code

```
                          /* ** begin win create window                                ** */
                         /* Create the WC_ENTRYFIELD, ""  */
my entry field           call WinCreateWindow(hwnd0, WC_ENTRYFIELD->szBased, "",
                                      ior( WS_VISIBLE ,  ES_MARGIN), 152, 80, 44, 24,
                                      hwnd0, HWND_TOP, EF_ID, null(), null());
                          /* ** begin set entry field limit                           ** */
- length=5               /* Max no. of chars for EF_ID is 5 */
                         call WinSendDlgItemMsg(hwnd0, EF_ID, EM_SETTEXTLIMIT,
                                           MPFROMSHORT(5), null() );
                          /* ** begin win create window                               ** */
                         /* Create the WC_BUTTON, "Retrieve"  */
my push button           call WinCreateWindow(hwnd0, WC_BUTTON->szBased, "Retrieve",
                                      ior(BS_PUSHBUTTON ,  WS_VISIBLE),
                                      64, 28, 97, 30, hwnd0, HWND_TOP,
                                      PB_RETRIEVE, null(), null());
                                 end; /* of when */
                    when (WM_PAINT)
- window painting    do;
                          /* ** begin default wm_paint                                ** */
                        /* Default code for WM_PAINT */
                        do;
                        ColorTable = 0 ;
                        hps0 = WinBeginPaint( hwnd0, null(), addr( rectl0 ) );
                        clr=SYSCLR_WINDOW;  /* default color for client */
                                            /* get background pres param if exists */
                        call WinQueryPresParam(hwnd0, PP_BACKGROUNDCOLORINDEX,
                               PP_BACKGROUNDCOLOR, attrfound, 4, addr(clr), QPF_ID1COLORINDEX);
                                          /* Change color table to accept RGB values */
                        call GpiCreateLogColorTable(hps0,0,LCOLF_RGB,0,0,ColorTable) ;
                        call WinFillRect( hps0,       /* paint the rectangle */
                                     addr(rectl0), clr);
                        call WinEndPaint( hps0 );
                        end; /* of default code for WM_PAINT */
                        end; /* of when */
                    otherwise
                        return(WinDefWindowProc(hwnd0,msg0,mp1,mp2));
* end of message    end; /* of select(msg0) */
  selection         return(ptrvalue(FALSE));
end; /* of procedure */
end; /* of package */
```

Figure 25 (Part 3 of 3). Visual PL/I Generated Code

A typical layout of the source code for any PM program is:

- Declare and include standard definitions.
- Initialize PM using WinInitialize; this returns an anchor block handle.
- Create message queue. PM programs communicate with each other mainly by sending and receiving messages from one another. Having initialized and received an anchor-block handle, WinCreateMsgQueue defines a message queue for this thread. It returns a message queue handle for a particular anchor block handle.
- Register the window procedure handling the main window.
- Perform the message loop. The window procedure will be called from within this loop.
- Destroy the queue and the window when the application ends.

Within the window procedure you will find a select statement that has three basic sections:

- The control section (WM_CONTROL), which contains the events. This is where our own code will be inserted. See “Code Blocks or Adding the Application Code” on page 43.
- The create section (WM_CREATE), which contains the definitions of all controls on the window; in our case the text field, the entry field, and the push button.
- The paint section (WM_PAINT), which contains the code to draw the window.

For people new to PM programming, using Visual PL/I and looking at the generated code will greatly enhance your understanding of PM programming.

Basic Window Design Techniques

The basic visual design techniques are:

Move Select a control with the mouse, hold down the left mouse button, and move the field around.

Copy Select a control, hold down the left mouse button and the Ctrl key, and move the cursor. Release the mouse button where you want the duplicate. If you release the mouse button without moving first, you cannot see the duplicate control!

Aligning and sizing multiple controls
Use the pop-up menu on each control and select Mark in Layout; this creates a group of controls. Then use the pop-up menu on one control and select Align or Size in Layout; the effect is on the whole group of marked controls. This is time consuming when you have many controls.

You can also display a grid (use the Project Settings notebook, Figure 19 on page 37) and snap new controls to the grid for alignment.

In general, create one control of each type with the proper attributes (select Styles from the pop-up menu of the control), and then use copy and group alignment.

Code Blocks or Adding the Application Code

Your own coding is added to the Visual PL/I application as code blocks using links. Use the pop-up menu of the project, window, or control to select the Links dialog. Links can be added at four different places:

Project Project level code blocks usually define global variables, common subroutines, and compile options (%process statement).

Window Window level code blocks can be used to initialize processing for one window, or to process messages the window receives from its controls.

Control Control level code blocks can be attached to any control of the window, for example, entry fields, radio buttons, spin buttons, and push buttons. These code blocks can change the behavior of the controls.

Menu Menu level code blocks can be attached to the items of an action bar or pop-up menu. See “Adding an Action Bar” on page 54 for an example.

Figure 26 on page 44 shows the Links dialog for the project.

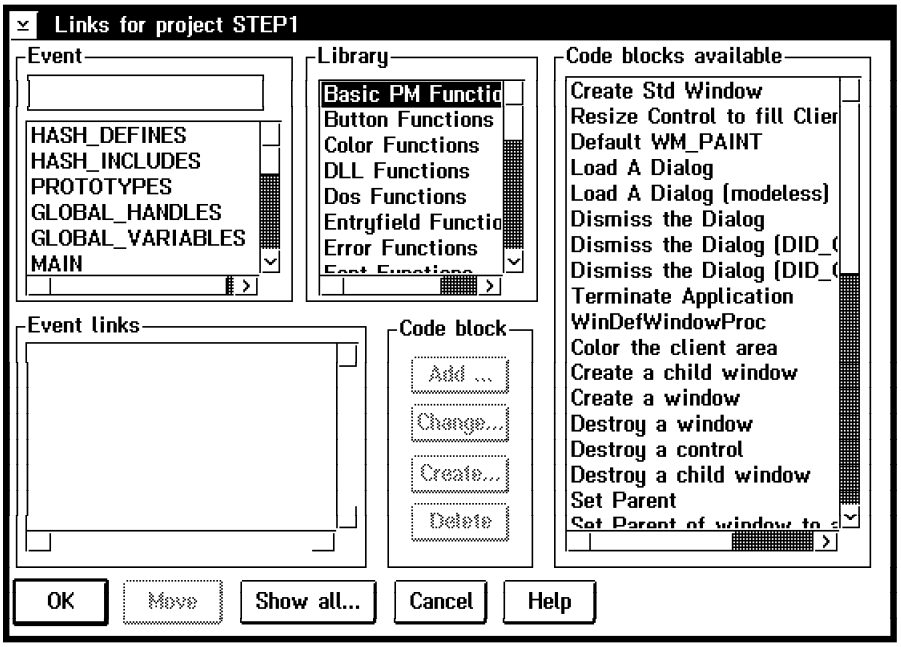


Figure 26. Links at the Project Level

The dialog has five distinct areas:

Event Contains all of the events (or actions) to which code blocks can be attached.

Library Contains the different types of code blocks. Typical types of code blocks are:

- Basic PM Functions (create window)
- Button Functions (click, enable, check)
- Entryfield Functions (read, set)
- PL/I Code (DO, IF, global variables, Any code)
- SQL Functions (host variables, include SQLCA, connect to database, commit, cursor functions (open, fetch), *process pp(sql), SQL statement, Any SQL code)
- My Code Blocks (empty). This is the place for the reusable code you develop.

Code blocks available
Lists all of the predefined code blocks of the selected library.

Event links In this area you build a list of code blocks that must be executed for the selected event. This is your application logic.

Code block In this area you select the action, for example, add, change, or delete a new code block to the Event links list.

When you attach code blocks to windows or individual controls, the list in the Event area changes; everything else remains the same. Figure 27 on page 45 shows the list of events for a window and a push button. Note how the list of events changes.

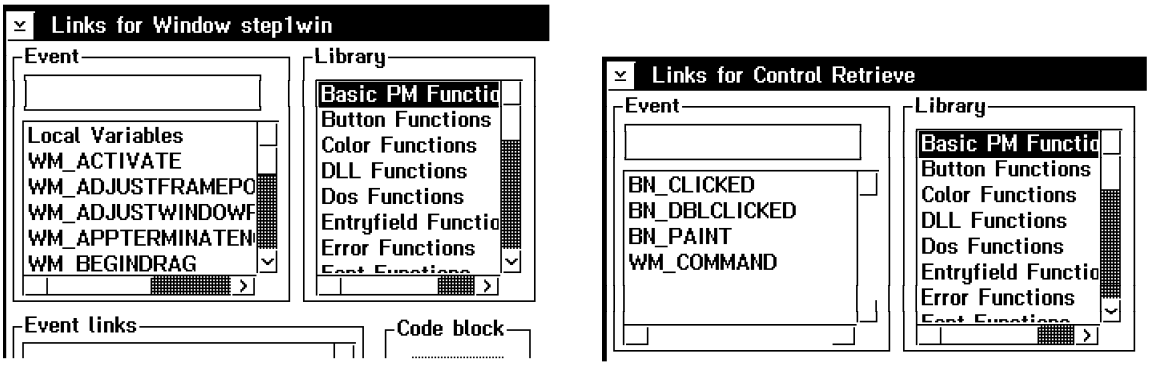


Figure 27. Links at the Window and Push Button Level

The Meaning of Events

To get a better idea of what an event means, double-click on an Event entry and you get a short help description about the event.

For more information on events consult the Presentation Manager literature mentioned in “Related Publications” on page xx.

Code Block Example

Suppose you want to open a secondary window when you click on the Retrieve push button in the STEP 1 Main Window (Figure 23 on page 39).

First define the secondary window; name it, for example, STEP1SUB.

Use the pop-up menu of the Retrieve push button to open a links window. Select the WM_COMMAND event (this means the button is clicked), the Basic PM Functions library, the Create a window code block, and the Add action. A list of all secondary windows of the application appears, you select STEP1SUB, and the code block is added to the Event links area (see Figure 28 on page 46).

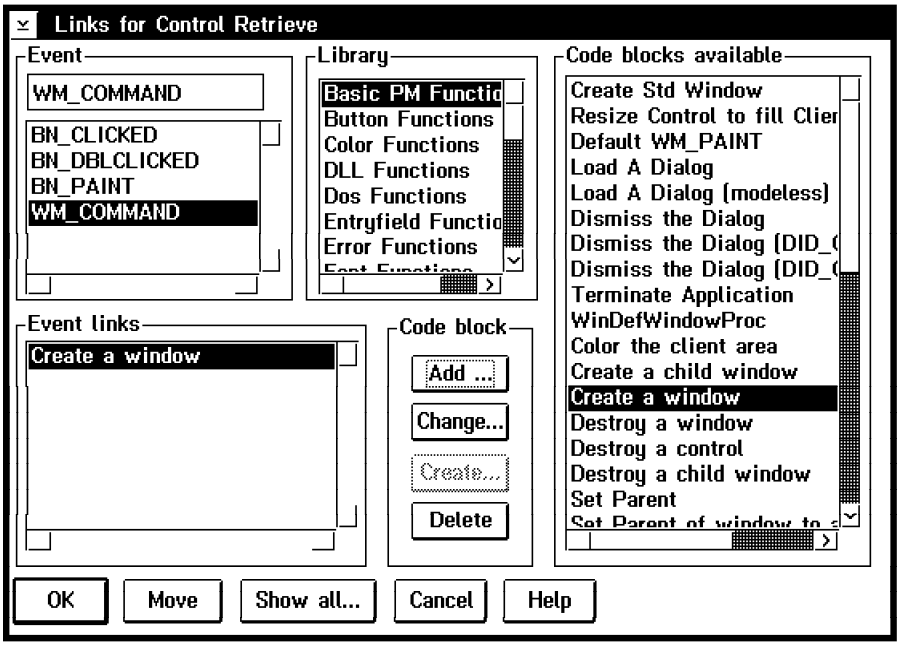


Figure 28. Adding a Code Block to a Push Button

To have the entry field value (personal number) available in the secondary window you would add code blocks to retrieve the entry field (Read Entry Field code block in the Entryfield Functions library) and store it in a global variable (Any code code block in the PL/I Code library). The global variable would be defined at the project level in the GLOBAL_VARIABLES event as a GLOBAL Char Variable of the PL/I Code library.

This small example should give you a basic understanding of how application code is attached in small pieces (code blocks) to PM events.

Viewing the PL/I Code of a Code Block

To view the actual PL/I code of a code block double-click on the entry in the Event links area. Viewing the code can give you insight into the actual PM calls.

Creating Useful Code Blocks

In your first Visual PL/I application you will have a tendency to write most of the logic in code blocks of the PL/I Any code type.

It is important to design applications such that you can create reusable code blocks. Instead of writing much of the logic in Any code blocks, you want to create your own application-specific library of reusable code blocks.

In this section we present examples of reusable code blocks.

My Code Blocks

Select My Code Blocks in the Library area of any Links dialog. The list of available code blocks is empty. The Create button (lower middle) is now active, and you can create your own code block and give it a useful name.

My code blocks are stored in the PMGMYCB.PLF file in d:\PLITK. They are loaded automatically when Visual PL/I is started. In “Creating a Library of Code Blocks” on page 49 we describe how to create application-specific reusable code blocks.

Example of My Code Blocks

Suppose you write a CICS transaction using Visual PL/I. You need to invoke the CICS preprocessor when compiling the generated code. The easiest way to invoke the preprocessor is through a *process statement at the beginning of the program.

Create a one-line code block named CICS Preprocessor as shown in Figure 29.

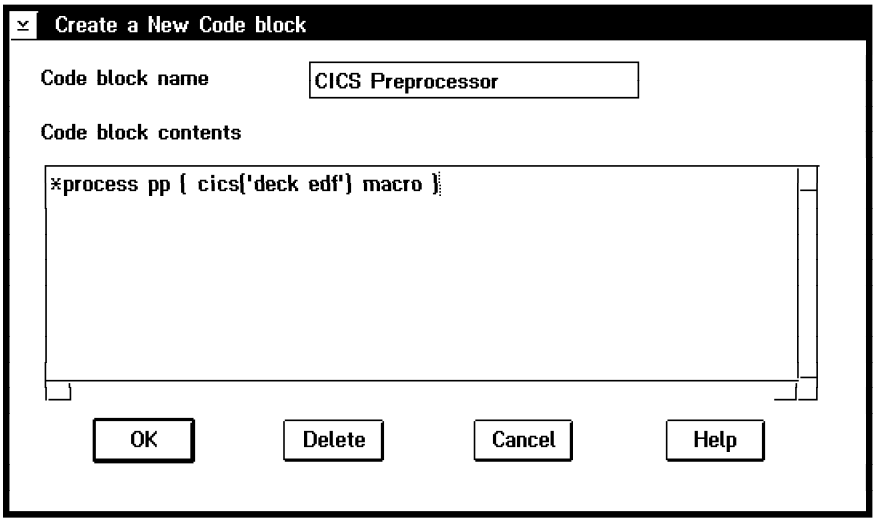


Figure 29. Creating a Code Block for the CICS Preprocessor

Now you have to add the CICS preprocessor code block to the beginning of the program. Use the pop-up menu of the project to open the Links dialog. Select the first event (HASH_DEFINES), the My Code Blocks library, the CICS Preprocessor code block, and Add.

The preprocessor is added to the Event links area after the predefined event link (# defines). See Figure 30 on page 48.

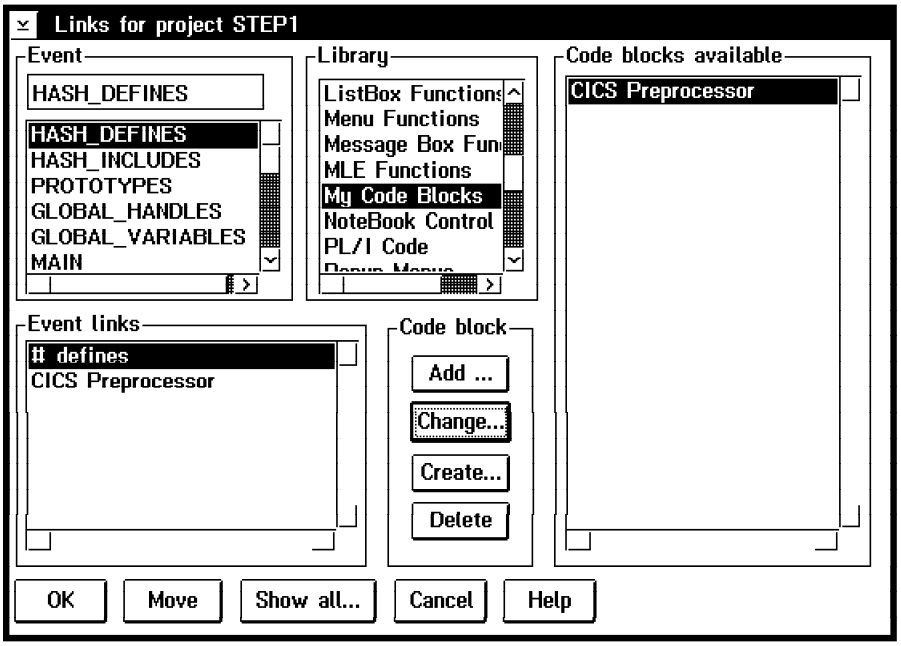


Figure 30. Using the Code Block for the CICS Preprocessor

To move the CICS preprocessor to the very top of the Event links list, select the # defines entry and click on the Move push button.

Advantages of My Code Blocks

Use the My Code Blocks library rather than Any Code of the PL/I Code library, if possible, for the following reasons:

- My Code Blocks can have their own meaningful title (such as “Validate user input against database”).
- Any Code will always appear as “Any Code” in the Event links list. It is far easier to find self-explanatory titles in the list.
- My Code Blocks can be easily moved and replicated, whereas it is difficult to reuse Any Code. You need to copy Any Code using an editor. My Code Blocks are particularly handy when you want to test and debug a small portion of the code, perhaps even in a separate project.

The only drawback with using My Code Blocks is when you are moving your applications from one workstation to another. The whole application is included in the master .PMG file of the project, except for the My Code Blocks; therefore you must move the code blocks file separately. All of the PL/I code of My Code Blocks is stored in one file, PMGMYCB.PLF, which is automatically loaded when Visual PL/I is started.

My Code Blocks are very useful for generic global code blocks for all applications. There is also a requirement for reusable code within one application.

Creating a Library of Code Blocks

The My Code Blocks library will grow quickly as you develop more applications. Visual PL/I allows for multiple libraries of code blocks as described in Appendix B, “Creating Your Own Code Blocks,” in the *PL/I for OS/2 Toolkit Reference*.

Code Block Converter

We wrote a small REXX program, MYCB2CB.CMD, which automates the task of creating a separate code block library. Refer to “REXX Program to Generate Own Code Blocks” on page 119 for the source code.

This program reads all of the code blocks from My Code Blocks (PMGMYCB.PLF) and creates the four files necessary for a separate application library of code blocks with a user-assigned name.

The advantages of using application-specific code block libraries are:

- Once you have created the code blocks you may not want to use them for every project. MYCB2CB.CMD lets you create meaningful libraries of code blocks for one application.
- You can easily “export” your tested code blocks to other developers. My Code Block is not as easily transported to other Visual PL/I machines.
- For very productive code developers, the PMGMYCB.PLF file will grow quickly, and MYCB2CB.CMD can be used as a housekeeping tool. Visual PL/I has a 30KB limit on the size of a code block.

The four files that make up a separate library of code blocks are:

- Program library file (name.PL), which contains the names of the other three files
- Module file (name.PLM), which contains the title
- Object file (name.PLO), which contains an identifier
- Function file (name.PLF), which contains the actual code blocks separated by BEGIN lines with the identifier (in name.PLO).

Figure 31 on page 50 shows the contents of and the relationships among the four files.

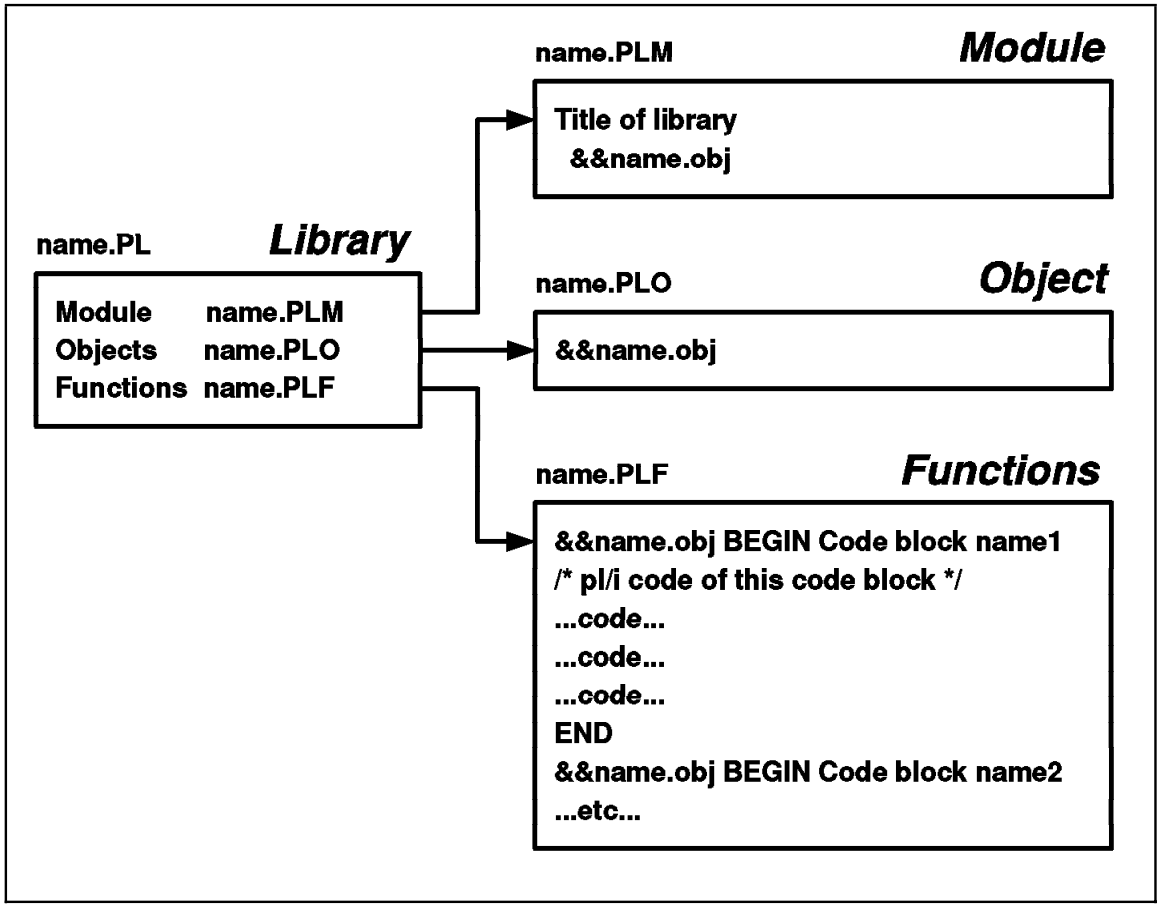


Figure 31. Code Block Library Files

Note: Figure 31 shows the simplest case of a code block library. For information on advanced code block libraries see the *PL/I for OS/2 Toolkit Reference*.

After generating the code block library, you need to rename the PMGMYCB.PLF file or delete the code blocks within that file; otherwise you will have duplicate code blocks.

Using Your Code Block Library

Before opening a project, read the contents of your code block library by selecting Read Codeblocks from the pop-up menu of the Visual PL/I for OS/2 folder (see Figure 18 on page 36). You are prompted for the name of the library file (name.PL).

Using Existing Source for a Code Block Library

There are some pitfalls when you copy existing source code into a code block library. Visual PL/I parses the text for keywords.

Visual PL/I Keywords

Visual PL/I keywords are always in uppercase. When copying (existing) source code into Visual PL/I code blocks, some of the text may actually be Visual PL/I keywords if it is in uppercase. For example, the line

```
CSQRETCD = STRING(PSFB) ;
```

contains the PL/I built-in function, STRING, which is also a Visual PL/I keyword. When Visual PL/I scans this line of code, it will interpret STRING as a Visual PL/I keyword instead of source code text. Thus, the generated code may be incorrect, or you may encounter syntax errors or abends when adding the code block to your Event links list. The two approaches to solving this problem are explained below.

Modify the Text: You can change STRING to lowercase or recode the text, depending on the circumstances. To convert a marked block of text to lowercase in the Enhanced Editor (of OS/2), use Ctrl+F4. This approach is not always possible, however. For example, if you are checking a variable value:

```
IF FIELD1 = 'STRING' THEN CALL ...
```

you cannot change STRING to lowercase. In this instance you could recode the statement as:

```
if field1 = 'STR'||'ING' then call ...
```

Use KEYWORDSOFF and KEYWORDSON: You can use the Visual PL/I KEYWORDSOFF and KEYWORDSON keywords around your source code to disable the parsing.

Note: Always browse the code blocks you have created yourself by adding them to an Event links list. Double-click on each line to view the code. You will save a lot of “head scratching” later on when the program is not performing as expected.

Productivity through Reuse

Much productivity can be gained through reuse. Visual PL/I is designed to make reuse easy through code blocks and its simple methods of copying existing objects, such as windows.

Reusing Existing Windows

Visual PL/I facilitates reuse of existing windows. You can:

- Copy a window within a project
- Copy a window from one project to another project
- Copy a window from another existing application not necessarily created using Visual PL/I.

The first two reuse methods are covered in Chapter 4 (“Copying Objects Between Projects”), Chapter 5 (“Copying Windows”), and Chapter 6 (“Copying a Menu”) in the *PL/I for OS/2 Toolkit Reference*. The last method is demonstrated in the PL/I for OS/2 promotion video.

Copying a Window of an Existing Application

To copy a window of an existing application, start that application to have the window available. For this demonstration we started the Seek and Scan Files application of the Productivity folder of OS/2.

In the Visual PL/I project, select Import and then Copy other window from the pop-up menu (see Figure 32).

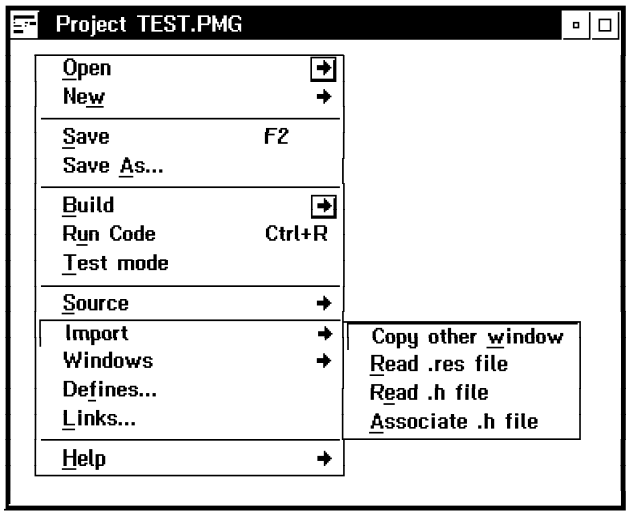


Figure 32. Importing an Existing Window

Select the existing application window with the mouse. Select OK in the confirmation window (see Figure 33).

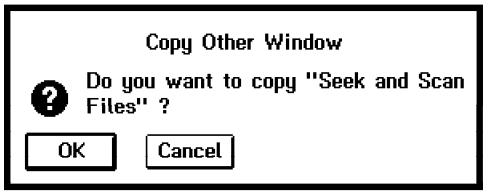


Figure 33. Confirmation Window

Now the project window has both a copy of the menu (action bar) and the window (see Figure 34).

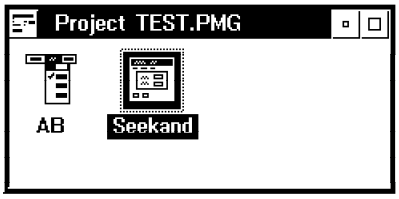


Figure 34. Project Window with Copy of Seek and Scan Files

Open the menu and the window to see how Visual PL/I re-created all of the controls as objects.

Application Design

In this section we look at some basic issues of designing PM applications with Visual PL/I.

In the previous sections we learned the basics of designing the first window of an application. Now we want to use some other facilities of PM to enhance the initial design.

Customizing Visual PL/I Objects

Through the Objects Manager you can set the default parameters for most of the controls. The settings that can be saved are:

- Title
- Initial width
- Initial height
- Initial style.

Visual PL/I objects are displayed in their own window as shown in Figure 20 on page 37. To change the defaults of a control, double-click on its icon. A Default Parameters window appears as shown in Figure 35.

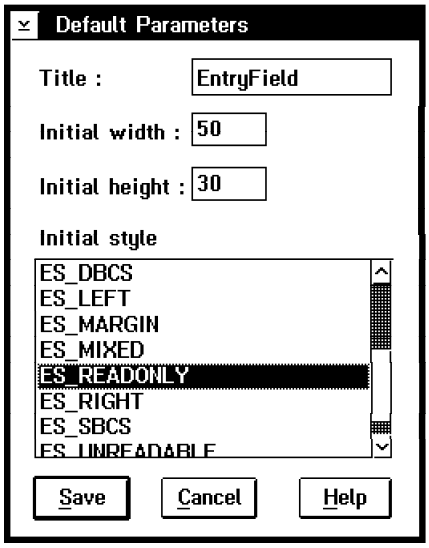


Figure 35. Setting Default Parameters for Controls

Say you were designing an output window with many read-only fields. Rather than going into each field to change the style to read-only, you can change the defaults for entry fields and save keystrokes.

Note: Most of the controls can be tailored, but not all of them will react to double-clicking.

Entry Fields

When designing entry fields be sure to check the Styles selection in the pop-up menu of the entry field. Decide on alignment (left, center, right), autoscrolling for long values (ES_AUTOSCROLL), and tabulator function on the window (WS_TABSTOP) and mark the field read-only (ES_READONLY) if it is used only for output.

Once you have created a field with proper attributes (styles) copy the field to add additional fields with identical characteristics. (Remember that you copy by moving a field with the Ctrl key pressed.)

Setting the Cursor

Only one field can have the cursor when a window is opened; by default the cursor is nowhere.

To set the cursor in an entry field use the pop-up menu of the window containing the field:

- Select the WM_CREATE event (window creation)
- Select the Set Focus to control code block from the Basic PM Functions library
- Use the Add push button and select the entry field from the list of controls that is displayed in a selection window.

Validation

In many applications input needs to be validated. We created a reusable code block (Check if numeric) for numeric validation of any entry field.

To make the code block reusable the coding uses a Visual PL/I variable name:

```
if WC_ENTRYFIELD VARNAME = ''  |
   verify(WC_ENTRYFIELD VARNAME,' 0123456789') ¬= 0 then
```

When such a code block is used Visual PL/I prompts you for the variable name.

Here is a typical coding sequence using the numeric validation:

```
Any code                      Message to blank
Read entry field              the field to validate
Check if numeric              our code block
 DO statement                 from PL/I Code
   Any code                   set error message
 END statement                from PL/I Code
Else statement                from PL/I Code
 DO statement
   normal processing          ... other code blocks
   Any code                   set good message
 END statement
Set entry field               display message
```

Adding an Action Bar

The action bar is called a menu in Visual PL/I. To create an action bar use the pop-up menu in the project window and select Menu in the New action.

Add each item of the action bar, with indented items for the pull-downs. See Figure 36 on page 55. Use the tilde (~) to mark the character to be underscored.

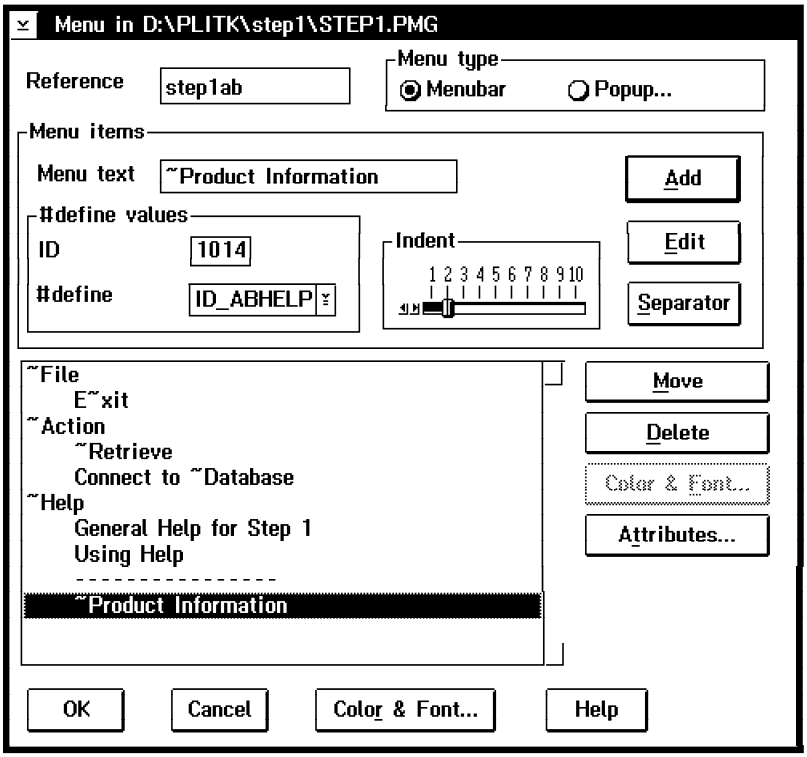


Figure 36. Creating an Action Bar

When you close this window (by clicking on OK), the action bar appears in the project as a separate icon. See Figure 37.

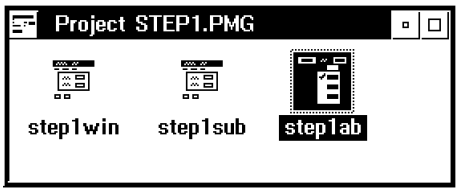


Figure 37. Project Window with Windows and Menu

The next step is to connect the action bar (menu) to the window. Use the pop-up menu of the window, select Change, and add the name of the action bar in the Associated action bar field.

Now when you open the window (by double-clicking on its icon) the action bar appears and you can select the items as shown in Figure 38 on page 56.

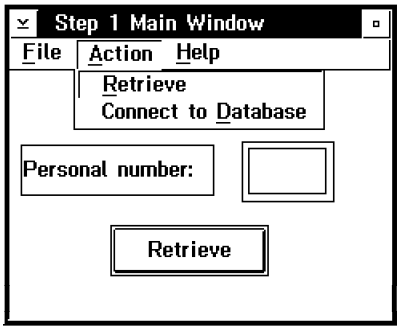


Figure 38. Application Window with Action Bar

Associating Code Blocks with Action Bar Items

To add logic (code blocks) to an item in the action bar use the pop-up menu of the action bar (menu) icon and select Links.

The events in the Links dialog are the menu items, and you can associate logic with each item as shown in Figure 39.

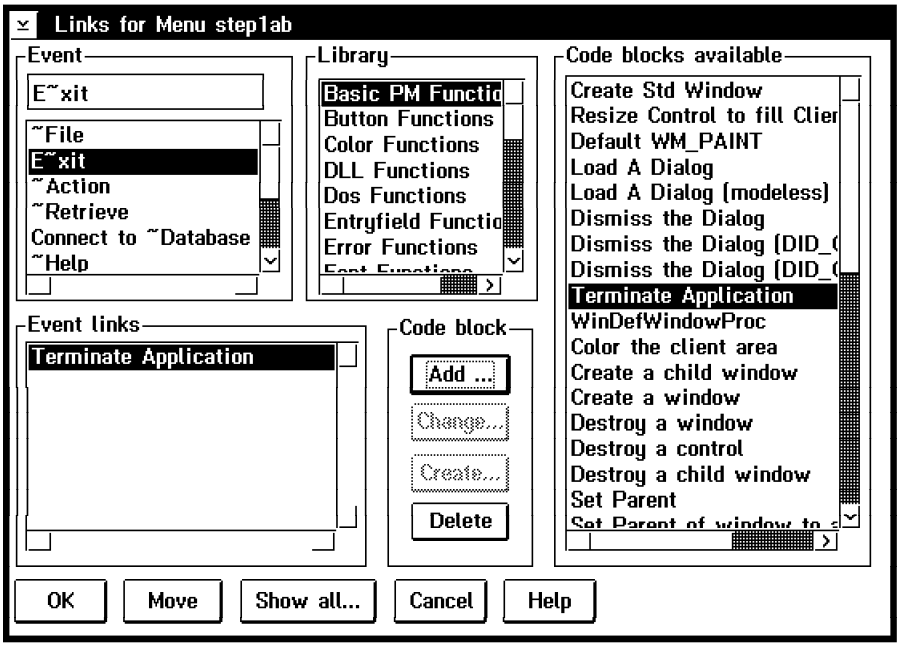


Figure 39. Application Logic for Action Bar Items

Adding Help

For real PM applications we need to provide typical help functions. PM, in conjunction with IPF/2, has different ways of presenting help text. To comply with CUA, refer to Chapter 8, “Common User Access Interface Components” in *Object-Oriented Interface Design, IBM Common User Access Guidelines*.

The standard ways of providing help are to:

- Activate function key F1
- Create a Help push button
- Create a Help action bar item with pull-down items (see Figure 36 on page 55).

Initializing IPF/2

You need to initialize IPF/2 before you can use the PM help facility:

1. Use the pop-up menu in the project window and select Links.
2. Select the SET_WINDOW_POS event, the Help Manager Functions library, the Initialize IPF/2 code block, and add the code block to the Event links area as shown in Figure 40.

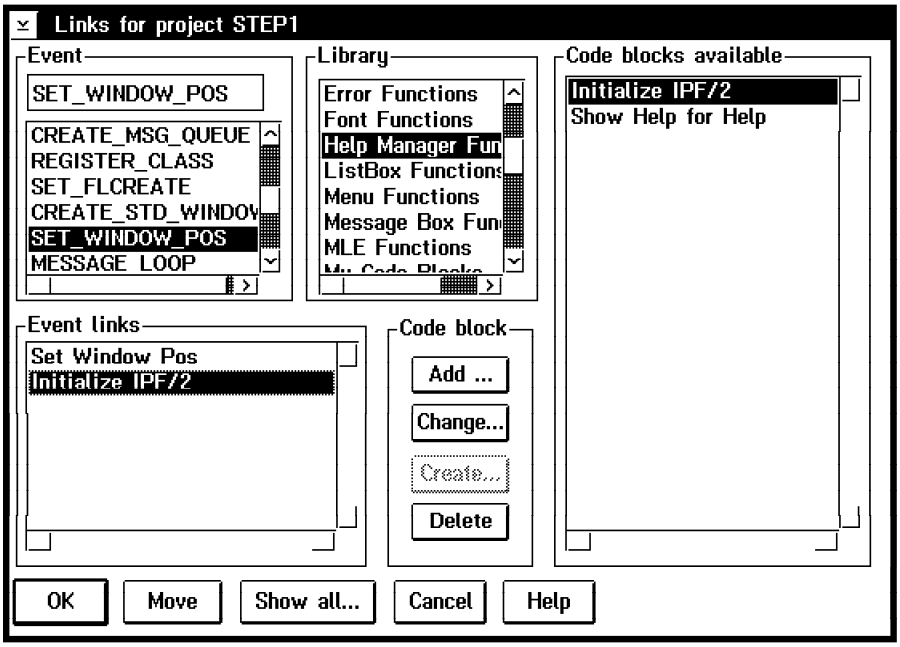


Figure 40. Initializing IPF/2 for Help

Help Text

Define the help text for the window by dragging the help icon (?) from the Visual PL/I objects (see Figure 20 on page 37) to the window.

A Help Information dialog window appears where you enter the help text as shown in Figure 41 on page 58.

Note: No indication that you defined help text is present in your project or window. To update the text, drag the help icon (?) from the Visual PL/I objects to the application window again.

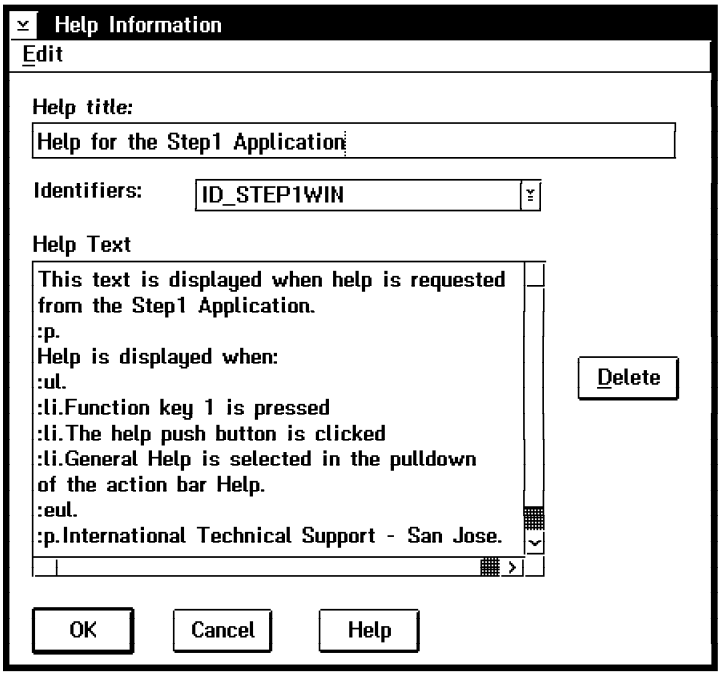


Figure 41. Defining the Help Text

Help text uses the IPF language. The simplest controls for paragraphs (:p) and lists (:ul., :li., :eul.) are shown in Figure 41.

Help Function Key

Initializing IPF/2 enables you to use the help key (F1) to activate help and display the help text.

Help Push Button

You can add a help push button to the window as a regular push button. Then select Styles from the pop-up menu and the BS_HELP item style.

Help on the Action Bar

Help for an application is usually added as an action bar item. You prepared the action bar with items associated with help (see Figure 36 on page 55).

To invoke the help text from the action bar you associate the help item, General Help for Step 1, with a code block as described in "Associating Code Blocks with Action Bar Items" on page 56.

Use a PL/I Any code block to call the standard help function:

```
call WinSendMsg(hwndOHelpInstance,
                HM_GENERAL_HELP,
                null(), null());
```

Product Information

Product information can be displayed as a subordinate dialog. Define a new window, Step1 Product Information, as shown in Figure 42.

Figure 42. Defining the Product Information Text

Use a window type of Dialog box and a line or dialog border. Also add an OK push button connected to the Dismiss the Dialog code block (from the Basic PM Functions library).

Connect the action bar item, Product Information (in the Help pull-down), with the Load a Dialog code block (from the Basic PM Functions library). See Figure 43.

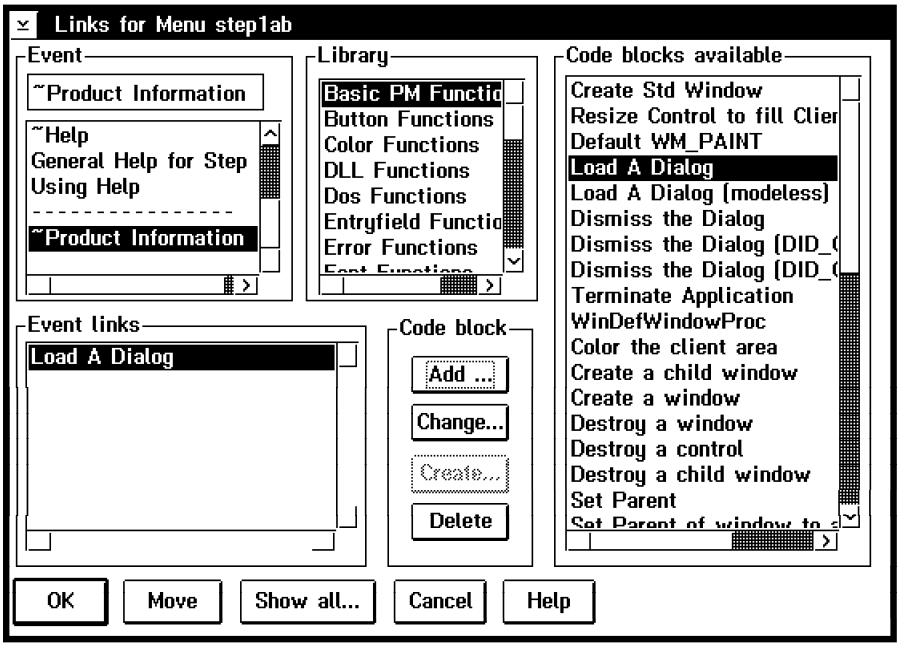


Figure 43. Event Link to Display Product Information

When you generate the application, help is enabled and available. You can use either the F1 function key, the Help push button, or the General Help action in the Help pull-down.

Adding the Application to the OS/2 Task List

Visual PL/I applications are not added automatically to the OS/2 task list. The OS/2 task list is displayed when you press Ctrl-Escape or click both the left and right mouse buttons on the desktop.

To add a Visual PL/I application to the task list select Styles in the pop-up menu of the main window and select the FCF_TASKLIST item style.

Note: If you minimize your Visual PL/I application, you can find it again in the Minimized Window Viewer folder even if it has not been added to the task list.

Application Termination

To terminate an application you can use three basic methods:

- Enable the system menu for the main window. You enable the system menu through the pop-up menu for the window and use either the Change action and check the system menu style or the Styles action and select FCF_SYSMENU.
- Define an End or Close push button and add the Terminate Application code block of the Basic PM Functions to the WM_COMMAND event (see Figure 39 on page 56).
- Use the OS/2 task list to terminate the process (see “Adding the Application to the OS/2 Task List”).

Calling an External Function

If you have an existing PL/I subroutine that you want to call from your Visual PL/I application, follow these rules:

- By default PL/I passes arguments by address, but Visual PL/I passes arguments by value. Therefore you must declare your external entry with the appropriate options:

```
DCL subrout ENTRY EXTERNAL OPTIONS(BYADDR);
```

- Update the Compile and Link options in the Visual PL/I Project Settings (Figure 50 on page 67):
  - Include the object code module name under Other object files
  - Include appropriate libraries if the subroutine uses SQL or CICS functions.

Changing Fonts and Colors

The easiest way to change the fonts and colors of windows and controls is by starting the Font Palette and Color Palette from the OS/2 System Setup folder.

Drag and drop any of the fonts and/or colors from those windows to a control (for example, a push button) in a Visual PL/I application window. The control changes its appearance and is defined in this appearance when the application is generated.

You can also start the Scheme Palette from the OS/2 System Setup folder. Using the Scheme Palette, you can drag a whole scheme (fonts and colors) to a Visual PL/I application window, and all of the controls in that window will change to the colors and fonts of the scheme.

Error Handling and Debugging

When you try to generate and compile the Visual PL/I code, you will encounter error situations quickly. In this section we present some guidelines for error handling and debugging.

Compilation Problems

Visual PL/I generates the source code first, including a make file, and then runs the make file to compile and link the application.

If the compile fails, you will get an error message as shown in Figure 44.

```
        pli STEP1.pli ( comp.options   > STEP1.err
NMAKE : fatal error U1077: 'C:\OS2\CMD.EXE' : return code '8'
Stop.
```

Figure 44. Build Failure in Compilation

A nonzero return code indicates that the compiler generated some warning or error messages. The log of running the make file is written to the STEP1.ERR file; the compiler output listing is in the STEP1.LST file.

You have to browse or edit those files to analyze the error messages and then relate the errors back to the design or logic of your Visual PL/I application. This is the reason why you have to understand the basic layout of a Visual PL/I program (see Figure 25 on page 40). In most cases the errors come from code you have added to events, and many times the code is in PL/I Any code code blocks.

The compile and link process stops even if there are only warnings, that is, a return code of 4. You can continue the build by selecting the arrow next to Build, and then Compile Code, in the pop-up menu of the project (see Figure 45).

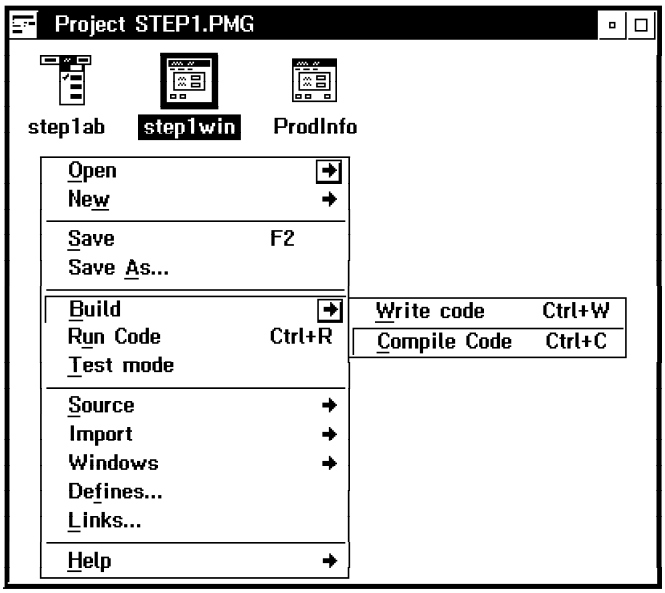


Figure 45. Continuing the Build Process

Debugging Visual PL/I Programs

Visual PL/I does not support the PLITEST facility of PL/I for OS/2. You have to debug your programs using more simple facilities, such as writing test output to files, capturing errors using conditions, and using the PLIDUMP facility.

Your PL/I manuals cover this topic in detail:

- Chapter 10, “Testing and debugging your programs” in the *PL/I for OS/2 Programming Guide*
- Chapter 16, “Condition handling” in the *PL/I for OS/2 Language Reference*
- Chapter 17, “Conditions” in the *PL/I for OS/2 Language Reference*.

Displaying Test Output

You can use the PL/I DISPLAY statement and write to the standard SYSPRINT file from a Visual PL/I application. At execution time Visual PL/I opens an appropriate window to display your output.

Useful functions are:

- DISPLAY, to write single line test output. Visual PL/I opens a Program Information Display window for the ouptut (see Figure 46).

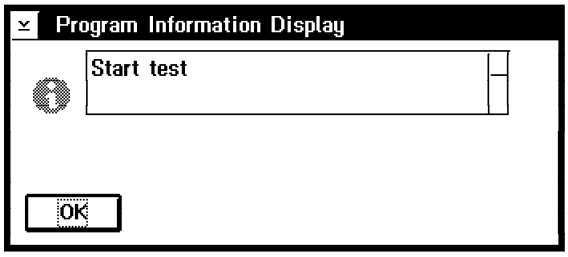


Figure 46. Visual PL/I Display Output

- PUT LIST, PUT DATA, and PUT EDIT statements, to write anything on SYSPRINT. Visual PL/I opens a SYSPRINT window where output is continuously displayed while your program is running.
- Built-in functions specific to PL/I for OS/2, such as PROCNAME, PACKAGENAME, and SOURCELINE.

  Remember to use the LANG(SAA2 NOEXT) compile option because these built-in functions are only for the OS/2 version of PL/I.
- ON ERROR condition, to capture most programming errors.
- ON ANYCONDITION, to capture any condition that may occur.
- PLIDUMP facility, for example, CALL PLIDUMP(’TFC’,”My dump’). Visual PL/I opens a PLIDUMP window to display the information.
- SNAP facility, to obtain a calling trace (a subset of PLIDUMP).

Note: Visual PL/I does not honor the file specifications for SYSPRINT and PLIDUMP. In normal PL/I for OS/2 applications you can specify:

```
set DD.sysprint = d:\directory\filename
set DD.plidump  = d:\directory\filename
```

Code Blocks for Debugging

Use the Visual PL/I code block facility to create reusable code blocks as shown in Table 2.

Table 2. Sample Code Blocks for Debugging

Function	Code Block
DISPLAY procedure and source line	`/* Display procedure/sourceline code block */` `display ( procname() || sourceline () ) ;`
DISPLAY text and variable	`/* Display a text and a variable code block */` `display ('VARIABLE "text to be displayed" '||VARIABLE "variable name");`
Put data of a variable	`/* Put data(variable) code block */` `PUT DATA ( VARIABLE "name of variable to display" ) ;`
CALL PLIDUMP	`/* call plidump code block */` `call plidump ( 'TFC', 'VARIABLE "Enter dump title"' ) ;`

ON CONVERSION	`/* On Conversion code block */`
	`on conversion begin ;`
	`  display ( 'Conversion raised in '|| procname() || sourceline () ) ;`
	`  display ( 'Onchar:  ' || onchar() ) ;`
	`  display ( 'Onsource: ' || onsource () ) ;`
	`end ;`

ON ERROR	`/* On Error code block */`
	`on error begin ;`
	`  on error system ;`
	`  call plidump ( 'TFC', 'On error dump:'||procname()||sourceline() );`
	`end ;`

The VARIABLE keyword in a code block triggers Visual PL/I to display a dialog with the text following the keyword (in double quotes) and prompt the user for a variable name. Such code blocks are more reusable than code blocks without variables because they can apply to any Visual PL/I program.

Compile and Link Options for Debugging

Use the following options for maximum debugging help:

GONUMBER Include statement numbers in run-time messages. You must also use the linkage editor option, /CO (codeview), to include the symbol table.

INTERRUPT Enable the Ctrl-Break key combination to stop your program and pass control to an ON ATTENTION condition. You must also use the INTERRUPT(ON) run-time option:

```
%process interrupt  runops('interrupt(on)')
```

SOURCE Include the source program in the compile listing.

TEST Invoke the PLITEST facility at runtime. Note that PLITEST is currently not supported for Visual PL/I programs.

Application Scenarios

In this section we show how you can implement the two applications described in Chapter 3, “Application Scenarios for PL/I for OS/2” on page 19 using Visual PL/I.

GUI Front End for ISPF/DB2 Application

The GUI front end for our MVS ISPF/DB2 application has two panels:

- A user panel for entering and displaying data (see Figure 11 on page 20)
- A help panel.

We want to use the subroutine accessing DB2 (SCLS102) unchanged. We replace the main program (SCLS101) with a Visual PL/I program and use the help facility of PM to display similar help text.

Project Layout

The project for the application consists of one window, a menu for the action bar, and a dialog for product information (see Figure 47).

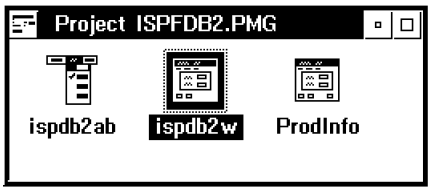


Figure 47. ISPF/DB2 Project

The action bar is taken directly from the STEP1 application (see Figure 37 on page 55) without the Action pull-down entry, Connect to Database.

Application Window Layout

Using all of the techniques of the STEP1 application, we designed the main window as shown in Figure 48 on page 65.

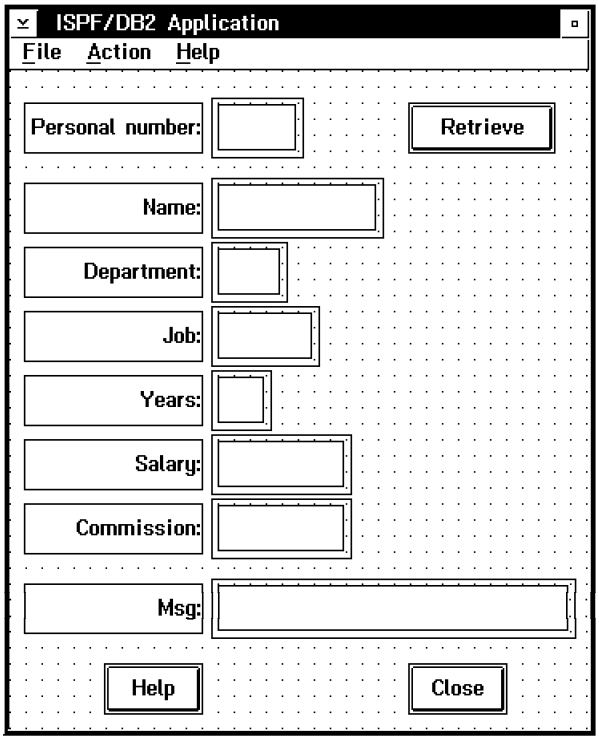


Figure 48. ISPF/DB2 Application Window Layout

We used a grid to easily align the text constants (right align) and entry fields (left align). The grid is defined using the Project Settings notebook where you can define the size of the grid and specify that all newly placed controls should be aligned (snapped to the grid).

The grid tab of the Project Settings notebook is visible in Figure 50 on page 67.

Application Interface to the DB2 Access Module

For each entry field a global variable is defined, such as visid, visname, visdept, and visjob. All of these are character variables. A global internal procedure, DB2CALL, is used to call the DB2 access module, SCLS102, and convert the results from their internal numerical format to the character format required for the entry fields.

The internal procedure, DB2CALL, is invoked from an event link attached to the Retrieve push button. The results are displayed using a Set entry field code block for each field.

Figure 49 on page 66 shows the code of the internal procedure to invoke the DB2 access module.

```
DB2CALL:  PROC;
/*********************************************************************/
/**   Visual PL/I Interface to SCLS102 on the workstation          **/
/*********************************************************************/
dcl scls102         entry external     options(byaddr),
    cid             bin fixed(15)      init(0),
    cname           char(9) var        init(' '),
    cdept           bin fixed(15)      init(0),
    cjob            char(5)            init(' '),
    cyears          bin fixed(15)      init(0),
    csalary         fixed decimal(7,2) init(0),
    ccomm           fixed decimal(7,2) init(0),
    csqlcode        bin fixed(31)      init(0);
dcl pic5            pic'99999',
    pic5z           pic'ZZZZ9',
    pic3            pic'999',
    pic6s           pic'SZZZZ9',
    pic8            pic'----9.V99';
/* logic */
    pic5 = visid;
    cid  = pic5;
    call SCLS102 ( cid, cname, cdept, cjob, cyears,
                        csalary, ccomm, csqlcode );
    if csqlcode = 0 then do;
        vismsg    = 'DB retrieve successful';
        visname   = cname;
        visjob    = cjob;
        pic3      = cdept;    visdept   = pic3;
        pic5z     = cyears;   visyears  = pic5z;
        pic8      = csalary;  vissalary = pic8;
        pic8      = ccomm;    viscomm   = pic8;
        end;
    else do;
        pic6s     = csqlcode;
        vismsg    = 'DB retrieve failed, rc='||pic6s;
        visname   = '--none--';
        visdept   = ' ';
        visjob    = ' ';
        visyears  = ' ';
        vissalary = ' ';
        viscomm   = ' ';
    end;
END DB2CALL;
```

Figure 49. Interface Code for DB2 Access Module

Compile and Link Options

We copied the database access module, SCLS102.OBJ, into the project directory and used compile and link options as shown in Figure 50 on page 67.

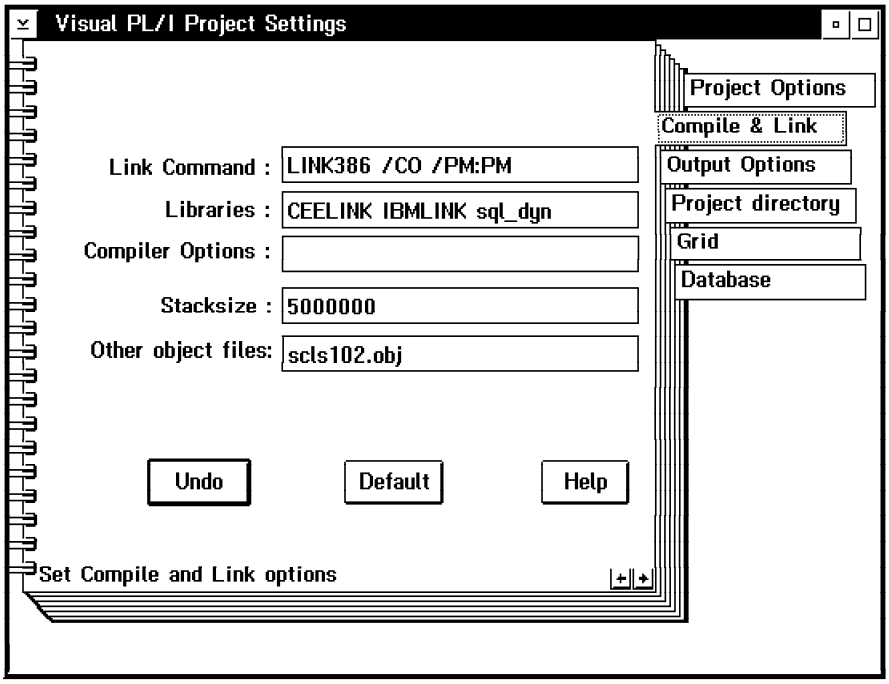


Figure 50. ISPF/DB2 Compile and Link Options

Alternatively we could have put the object module file into an OS/2 library using the lib utility of the IBM Developer's Toolkit for OS/2.

Running the Application

Make sure that DB2/2 is running (if not, use the STARTDBM command) and log on using a local userid.

You can start the application from the project pop-up menu (Run code) or in an OS/2 window.

When the main window is displayed, enter an ID, and select Retrieve. Figure 51 on page 68 shows a sample output.

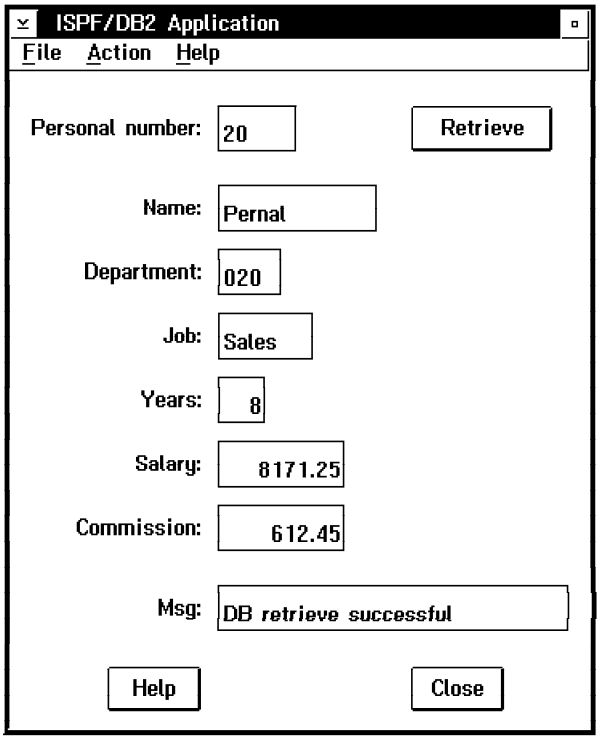


Figure 51. ISPF/DB2 Application Run

You can use the help key (F1) or Help pull-down to display the help text. Product Information is available under the Help pull-down. Figure 52 shows the resulting windows.

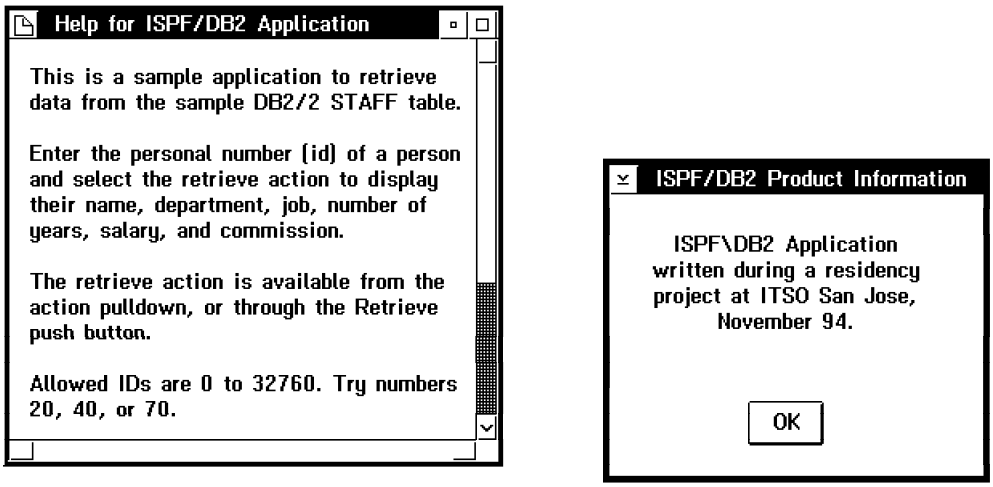


Figure 52. ISPF/DB2 Application Help and Product Information

GUI Front End for CICS/DB2 Application

We designed a GUI front end interface using Visual PL/I to replace PLIECI, the small test program described in “CICS External Call Interface” on page 29.

We invoked the CICS transaction programs, PCASH and PNEWOR3, through ECI calls from the program built using Visual PL/I.

This application shows the use of the CICS ECI from Visual PL/I as well as the retrieval of information from DB2 tables using SQL statements.

We did not develop help or product information for this application. The same techniques used for the ISPF/DB2 application could be applied.

Figure 53 shows the CICS ECI project folder.

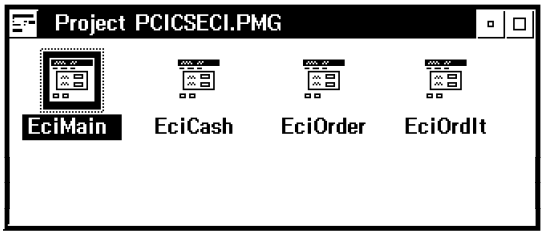


Figure 53. CICS ECI Project Folder

Window Design

The small GUI front end consists of four windows:

- Main window, to invoke either the cashier (PCASH) or order (PNEWOR3) transaction (see Figure 54 on page 72).
- Cashier window, to display the information retrieved from the PCASH program (see Figure 55 on page 72).
- Order window, to display the order information. This information is retrieved using DB2 calls from the ORDERT table. The PNEWOR3 transaction can then be invoked to update the order information (see Figure 56 on page 73).
- Order item window, to display the information of one item of the order. This information is retrieved from the ORDER_ITEM table. A subset of the denominations for that item from the ORDER_DENOM table is displayed as well (see Figure 57 on page 73).

Using the CICS ECI from Visual PL/I

The same code sequence as in the small test program, PLIECI, was used in Visual PL/I. The placement of the code fragments was as follows:

- The ECI parameter structure and the CICS communications area were defined in the global variable section of the Visual PL/I project:

```
/*********************************************************************/
/* Include the ECI control structures                                */
/*********************************************************************/
%include cics_eci;
dcl      eci_parms_s        type ECI_PARMS,
         sEciReturnCode     fixed bin(15),
         DSrc               fixed bin(31);
```

```
DCL   01 COMMAREA           CHAR(2000) INIT(' '),
         COMMAREA_PTR       PTR;
DCL   01 CASHIER_COM        BASED(COMMAREA_PTR) UNALIGNED,
         05 DB_DATA,
            %INCLUDE PCASHCOM;
         05 MESSAGES,
            %INCLUDE PMSGCOM;
DCL   01 ORDER_COM          BASED(COMMAREA_PTR) UNALIGNED,
         05 DB_DATA,
            %INCLUDE PORDCOM;
         05 MESSAGES,
            %INCLUDE PMSGCOM;
```

- The ECI call itself was placed into callable subroutines, which were also defined in the global section of the project. We developed one subroutine for PNEWOR3 and one for PCASH (as shown below).

```
/* PROC to call PCASH program using CICS ECI */
ecicash:  proc;
dcl vis_ecircp  pic'szzzz9';

eci_parms_s.eci_program_name = 'PCASH';
cashier_com.db_data.c_id   = vis_cashid;
cashier_com.db_data.c_name = '';
cashier_com.db_data.b_code = '';

sEciReturnCode = FaaExternalCall (eci_parms_s);

vis_cashname = cashier_com.db_data.c_name;
vis_branch   = cashier_com.db_data.b_code;
vis_pgmmsg   = cashier_com.messages.pgm_text;
vis_ecircp   = sEciReturnCode;
vis_ecirc    = vis_ecircp;
end ecicash;
```

- The ECI subroutines were invoked where needed as an action of a push button.

Using SQL Statements in Visual PL/I

Visual PL/I provides many predefined code blocks for SQL access. It is important, however, to properly set up the global definitions. In this sample application we used the following method:

1. Define basic SQL control structures in the links section of the only window using SQL calls, or alternatively in the project itself. We used SQL only in the order window; therefore we defined the SQL information in the links for that window under the Local Variables event.

   We used the following SQL code blocks:

   - Include the SQLCA, a predefined code block.
   - Include the host structures generated using DCLGEN. We used an “any SQL code” code block:

     ```
     EXEC SQL INCLUDE PORDER;
     EXEC SQL INCLUDE PORDITM;
     EXEC SQL INCLUDE PORDDEN;
     ```

   - A “Begin Hostvar Statement” code block to define additional host variables for the number of items in an order:

```
dcl horditems  fixed bin(31);
dcl horditem   fixed bin(31);
```

2. Connect to the database in the WM_CREATE event using the SQL Connect to database code block. Visual PL/I prompts for the name of the database, in our case, SRVDB.

3. Retrieve order information using an SQL SELECT statement. We used an “any SQL code” code block:

```
EXEC SQL
  SELECT * INTO :DCLORDER
           FROM SRV.ORDERT
           WHERE ORDER_NUM = :DCLORDER.ORDER_NUM;
```

4. Retrieve order items using a simple loop. Within the loop we used an SQL statement (any SQL code block):

```
exec sql SELECT *
         INTO :dclorder_item
         FROM srv.order_item
         WHERE order_num = :dclorder_item.order_num
           AND order_item_num = :dclorder_item.order_item_num;
```

5. Retrieve order denominations for an item using a cursor. The following code blocks are required:

   - Define the cursor (any SQL code block)

```
exec sql DECLARE cursden CURSOR FOR
  SELECT *
     FROM srv.order_denom
     WHERE order_num = :dclorder_denom.order_num
       AND order_item_num = :dclorder_denom.order_item_num;
```

   - SQL open cursor, a predefined code block.
   - A loop with an SQL fetch statement, a predefined code block. We fetched the data into the DCLORDER_DENOM host structure.
   - SQL close cursor, a predefined code block.

6. Count the number of order items by using an SQL statement (any SQL code block):

```
EXEC SQL
  select count(*)
         into :horditems
         from srv.order_item
         where order_num = :dclorder.order_num;
```

7. SQL commit statement, a predefined code block.

All of the code blocks to retrieve information from the tables were defined within the links of a Retrieve push button.

Many other code blocks were used between the SQL statements to move the information from SQL host variables to the CICS communications area and to variables used to display the information in entry fields in the window.

Compile and Link Options

The DB2 preprocessor must be invoked. In the links for the project, under the very first entry, HASH_DEFINES, a code block provided by Visual PL/I, PL/I *PROCESS PP(SQL Statement), is included with the name of the database (SRVDB). Use the Move push button in the Links window to make the *PROCESS code block the first code block for the event.

In the project definition the link libraries of the link options are defined as follows to include CICS and DB2 libraries:

```
CEELINK IBMLINK FAACIC32 SQL_DYN
```

Running the Application

Start CICS 2.0 first, and make sure that DB2/2 is running as well. Then start the PCICSECI application. The main window is displayed as shown in Figure 54.

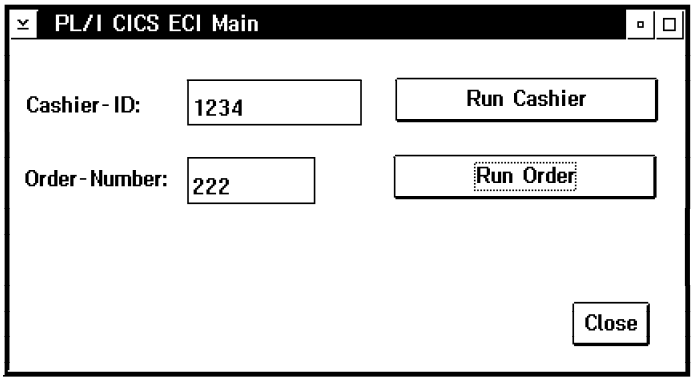


Figure 54. CICS ECI Main Window

Enter a cashier ID or order number and invoke the appropriate function using the push button.

For a cashier, the CICS transaction program, PCASH, is invoked immediately using a CICS ECI call. It retrieves the cashier information, which is then displayed in the cashier window (see Figure 55).

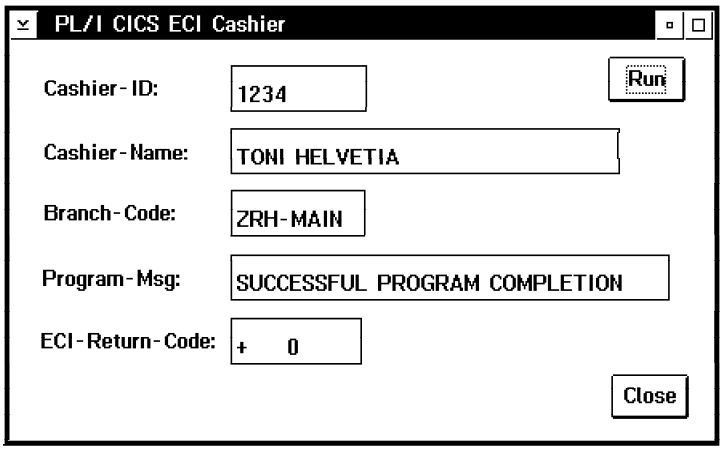


Figure 55. CICS ECI Cashier Window

For an order number, the order window is displayed and the order information can be retrieved from the ORDERT table in DB2/2 (see Figure 56 on page 73).

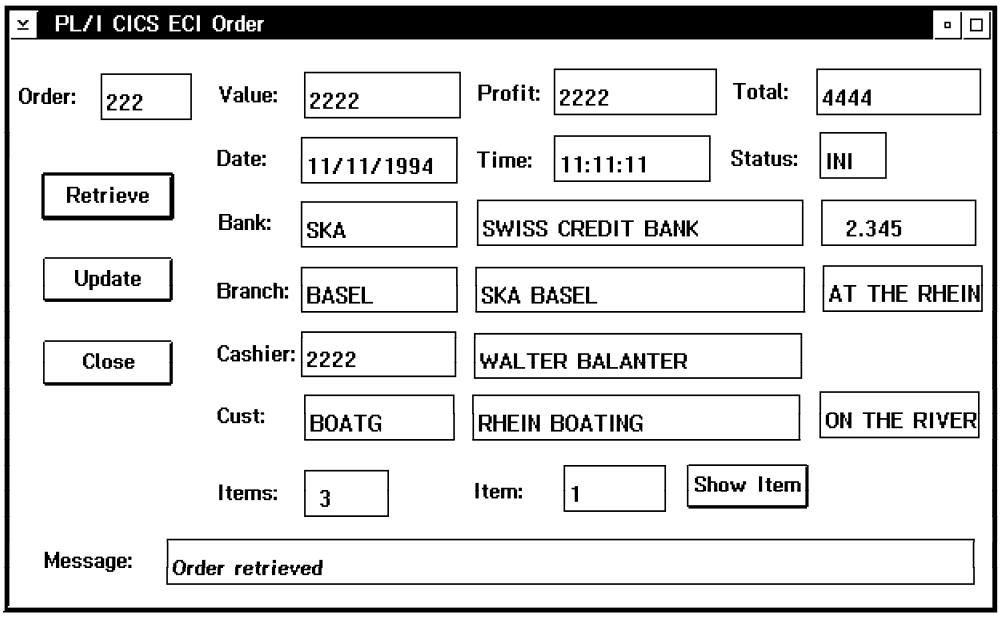


Figure 56. CICS ECI Order Window

Data can be changed in the window, and the CICS transaction program, PNEWOR3, is invoked using a CICS ECI call to update the tables.

You can also display an order item through the Show Item push button (see Figure 57).

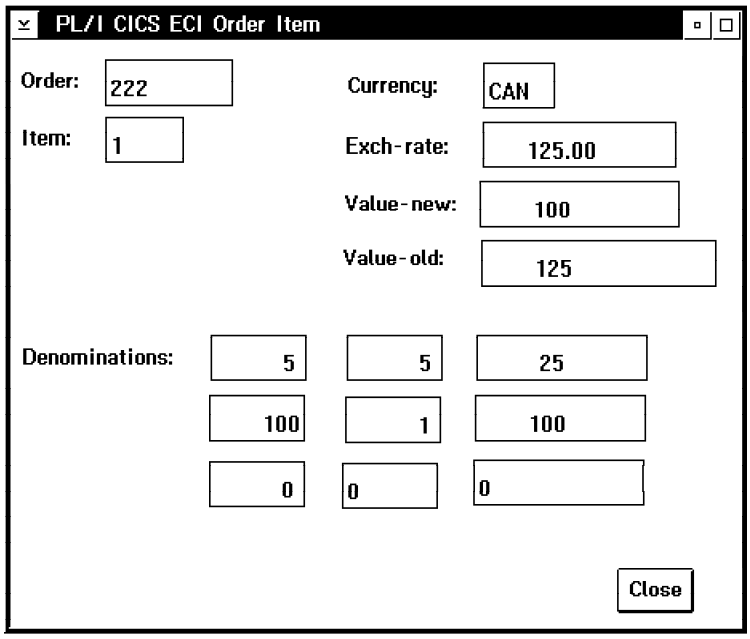


Figure 57. CICS ECI Order Item Window

This application uses an effective combination of Visual PL/I for the user interface, DB2/2 for the data, and CICS ECI for using existing programs.

Device-Independent Code

If you develop a Visual PL/I application on an Extended Graphics Adapter (XGA) or an SVGA system and then run it on a VGA system, your windows may be too large to fit on the display screen. Application windows developed on VGA will shrink, however, when run on an XGA or SVGA system, and data within fields may be truncated.

There is no easy solution to this problem. To solve it you might try to:

- Develop the application on a VGA system and make the windows as large as possible with extra empty space and overlong fields. When run on an XGA system, the windows will still look OK.
- Develop windows for different display adapters as separate DLLs and load the appropriate size dynamically.
- Dynamically size every window and potentially every control on the window according to the resolution in which you are running. This approach requires one extra code block for every control attached to the WM_CREATE event of the window.

  Note: A code block library (INDEVPLI PACKAGE) is available within IBM to help with these calculations. The code block library files are listed in “Visual PL/I Code Blocks for Device Independence” on page 138. These code blocks were converted from C to PL/I.

Instructions for Using Device-Independent Code Blocks

To create device-independent Visual PL/I applications follow these instructions.

Visual PL/I

- After starting Visual PL/I load the INDEVPLI.PL code block file.

Project Links

- Place the Global variables code block under the HASH_INCLUDES event.
- Place the Set Development Device code block under the SET_FLCREATE event. Make sure the block is positioned below the existing Set flCreate code block.
- Place the Set Window Position code block under the SET_WINDOW_POS event to set the correct size of the Primary Window. Delete the Set Window Pos link currently listed in the Event links area.

Window Links

- For all secondary window creation, use the Create Window code blocks instead of the code blocks supplied by Visual PL/I.
- Use the Set ″Control″ Position code blocks for the WM_CREATE event for all standard windows containing controls, such as push buttons, entry fields, and text constants (this is not necessary for dialog windows).

  You will need to add one code block for each control (push button, text constant, entry field, and so forth) under the WM_CREATE message for the Primary Window. Thus, when the Primary window is created, it will reposition and resize all of the controls.

Converting C Code to PL/I

Most PM applications are written in C. For customers who do not have the IBM C++ compiler installed, converting working C code (or code written in other programming languages) to PL/I is a way of building up a library of reusable code.

Converting C Header Files to PL/I

Visual PL/I comes with C2PLI.CMD, which converts C header files. Chapter 17, “Converting C Header Files to PL/I” in the *PL/I for OS/2 Toolkit Reference* deals with converting C header files.

After successful conversion, put the new .CPY files into a directory in the search path for include files (SET IBM.SYSLIB or SET INCLUDE).

Example: We ran the C code listed in Figure 58 through the C2PLI command.

```
/*------ C header file to convert -----------------*/

typedef long int myint;        /* integer          */
typedef long int *pmyint;      /* pointer to int   */
typedef char     mychar;       /* character        */

typedef struct _customer       /* customer         */
  {
   long int   custnumber;      /* - number         */
   char       name[20];        /* - name           */
   char       address[16];     /* - address        */
  } aCust;

int  _System custlist ( mychar   custsearch[20],
                        pmyint   numCusts,
                        aCust    customers[10] );
```

Figure 58. C to PL/I Conversion: C Header File (Input)

The conversion utility produced the PL/I header file shown in Figure 59 on page 76.

```
/*********************************************************
*   CREATED BY C2PLI CONVERSION UTILITY                  *
*********************************************************/
/*------ C header file to convert -----------------*/

define alias myint fixed bin(31);
define alias @myint pointer;
/* integer          */
define alias pmyint pointer;
define alias @pmyint pointer;
/* pointer to int   */
define alias mychar char;
define alias @mychar pointer;
/* character        */

define structure
  1 #customer,
   2 int custnumber fixed bin(31),/* - number */
   2 name char(20-1) varyingz,/* - name */
   2 address char(16-1) varyingz;/* - address */

define alias @#customer handle #customer;
define alias aCust type #customer;
define alias @aCust type @#customer;

dcl custlist  entry (
       type mychar  dim(0:20-1),
       type pmyint,
       type aCust  dim(0:10-1))
         returns( fixed bin(31) byvalue )
          options(linkage(system) byvalue nodescriptor) external;
```

Figure 59. C to PL/I Conversion: PL/I Header File (Generated)

Converting C Code Blocks to PL/I

Visual PL/I is adapted from an IBM internal tool called PM-Guide/2. PM-Guide/2 has code blocks written in C and generates executables or DLLs, similar to Visual PL/I.

We converted PM-Guide/2 C code blocks into Visual PL/I code blocks. PM-Guide/2 C code differs from straight C code in that it contains PM-Guide/2 keywords, most of which are also Visual PL/I keywords.

C is case sensitive. For example, `HWND` and `hwnd` are two different things in C. PL/I is not case sensitive, so when you convert from C to PL/I you must introduce new variables to distinguish uppercase and lowercase types. In our example we used `hwnd0` to distinguish the `hwnd` variable from `HWND`.

"Visual PL/I Code Blocks for Device Independence" on page 138 shows the result of converting a C code block library to a Visual PL/I code block library.

Hints and Tips for Visual PL/I Application Development

Here are a few ideas to help you exploit the power of PL/I for OS/2 and Visual PL/I and increase your own productivity.

- Shadow the Scheme, Color, and Font palettes of the OS/2 System Setup folder to the Visual PL/I folder. Dragging and dropping fonts and colors to Visual PL/I controls is a very intuitive and object-oriented way of performing window color and font tailoring.

  Make use of the grouping facility on controls to save keystrokes: “Working with a Group of Controls” on page 90 of the *PL/I for OS/2 Toolkit Reference* describes how to take advantage of this facility. Drop colors and/or fonts onto a group of controls.

- When creating a (primary) window, select the System menu check box from Window Styles in the Settings window. This selection enables you to close the window.

- Select the Minimize button check box from Window Styles to hide the window during execution.

- Add every (or at least the primary) window to the OS/2 task list (select FCF_TASKLIST in Styles).

- Make your most frequently used Visual PL/I code blocks appear at the top of the Library list of Links. For example, if you are frequently using My Code Blocks, rather than scrolling down the Library list each time, you can make My Code Blocks appear at the top of the list by updating the PMGUIDE.PLM file (in d:\PLITK):

  ```
  Default:                    Changed:
  ----------------------      ---------------------
  My Code Blocks              01My Code Blocks
   &&pmgmycb.obj               &&pmgmycb.obj
  ```

  Refer to “My Code Blocks” on page 47 for more time-saving tips.

- Use ON-units to handle exceptions and errors.

  To place ON-units inside the main procedure, add the ON-unit code block to the project links, under the MAIN event.

- Use the GONUMBER (GN) compile option and the /CODEVIEW (/co) link option. Together with PLIDUMP (which can be added as a code block to the application), these options will give you debug information. You cannot use the TEST compile option because PLITEST does not work for PM applications.

- Set compile options common to all projects in Visual PL/I settings and add project-specific compile options in the Project Settings.

- Use the lib utility of the OS/2 Developer’s Toolkit to put reusable object modules in an OS/2 library file (.LIB) to easily link them with your Visual PL/I applications.

Chapter 6. WorkFrame/2 and PL/I for OS/2

In this chapter we explore the use of WorkFrame/2 with PL/I application programs.

WorkFrame/2 is an optional component. You can compile and link PL/I programs using commands from an OS/2 window, as shown in Chapter 3, “Application Scenarios for PL/I for OS/2” on page 19. For Visual PL/I you do not need WorkFrame/2 because the compile and link process is automated as described in Chapter 5, “Visual PL/I of the PL/I for OS/2 Toolkit” on page 35.

WorkFrame/2 can be used effectively to manage your nonvisual PL/I applications, and that is what we want to explore here.

Installation and Setup of WorkFrame/2

As part of PL/I for OS/2 you receive the *PL/I for OS/2 WorkFrame/2 Guide*, which describes step by step how to install and set up WorkFrame/2.

WorkFrame/2 Folder and Objects

After you have installed WorkFrame/2 and applied the CSD 2 (or later) shipped with PL/I for OS/2, you will find a folder on your desktop (see Figure 60).

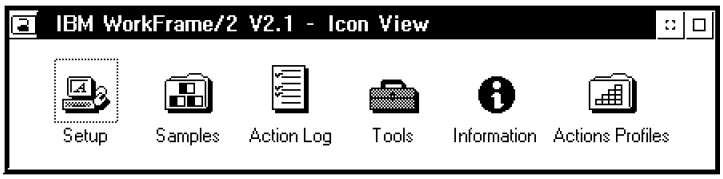


Figure 60. WorkFrame/2 Folder

The Setup icon can be used for some global settings of WorkFrame/2. We make use of the setup facility later (see Figure 71 on page 87).

Other important WorkFrame/2 objects are stored in the OS/2 Templates folder as shown in Figure 61.

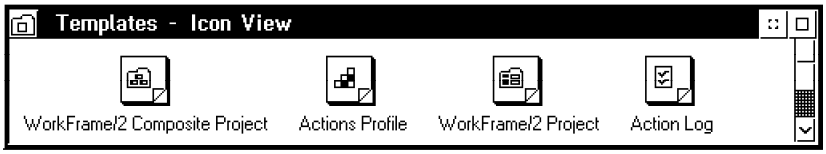


Figure 61. WorkFrame/2 Templates

Updating the Default Actions Profile

The CSD 2 shipped with PL/I for OS/2 adds a PL/I Actions Profile to the PL/I for OS/2 folder (see Figure 2 on page 7).

Open the PL/I Actions Profile and the WorkFrame/2 Default Actions Profile (in the Actions Profile folder).

Drag and drop the COMPILE action from the PL/I notebook to the Default notebook using the Ctrl key (copy operation). The other actions (edit, link, and make) seem to be identical in both notebooks.

Using WorkFrame/2 for PL/I Development

Let us explore how WorkFrame/2 can be used for your PL/I applications. The basic concepts of WorkFrame/2 are:

Project The project holds information about the source and target files of the application. It is linked to an Actions Profile and holds additional, project-specific compile and link options.

Actions Profile The Actions Profile holds all actions that can be applied to source and target files, for example, edit, compile, bind, and link.

Action Log The Action Log records all actions performed on files of the project.

Setting Up a Project

To explore the facilities of WorkFrame/2 we set up a project for the ISPF/DB2 application described in "ISPF/DB2 Application" on page 19.

The project is for the nonvisual version of the application, consisting of the reduced main program, SCLS101, and the DB2 access program, SCLS102.

The steps to set up a project are described in "Learning to Use WorkFrame/2 Version 2.1" in the *PL/I for OS/2 WorkFrame/2 Guide*. Here is an abbreviated version:

1. Open the OS/2 Templates folder (see Figure 61 on page 79).
2. Create a new folder on the desktop (drag the Folder icon from the Templates folder), open it, and change the name to your application, for example, WF/2 ISPF/DB2.
3. Drag and drop the Action Log icon from the Templates folder to the new folder.
4. Drag and drop the Actions Profile icon from the Templates folder to the new folder.
5. Drag and drop the WorkFrame/2 Project icon from the Templates folder to the new folder. The Project Settings notebook opens automatically; use the General page to change the name of the project to your application (ISPF/DB2 Project) and close the notebook.
6. Change the names of the Action Log and Actions Profile icons to your application (ISPF/DB2 Action Log and ISPF/DB2 Actions Profile).

Your new folder should now look like that in Figure 62 on page 81.

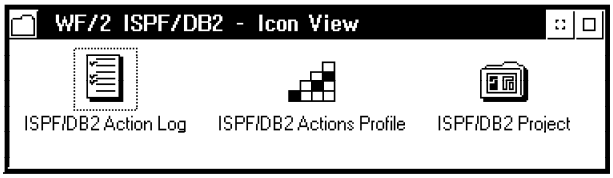


Figure 62. WorkFrame/2 ISPF/DB2 Project Folder

Tailoring the Project

Now we tailor the project for our application. Open the ISPF/DB2 Project - Settings notebook:

Source Set the source directory to where your programs are (d:\PLIDATA in our case), and the file mask to SCLS10*.*.

Target Set the make file name to SCLS101.MAK and the program name to SCLS101.EXE.

Profile Set the Actions Profile to the icon we created in the folder (ISPF/DB2 Actions Profile). Use the Find push button and look for the folder.

Figure 63 shows one page of the ISPF/DB2 Project - Settings notebook.

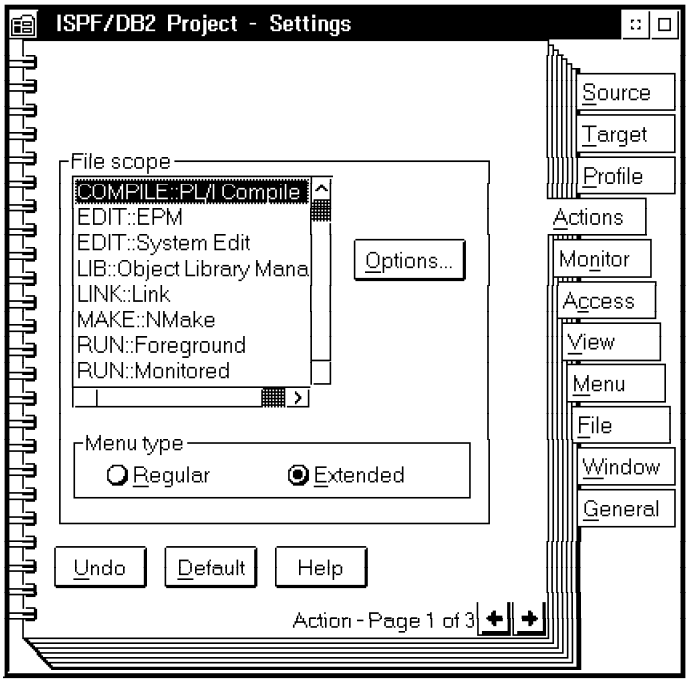


Figure 63. WorkFrame/2 ISPF/DB2 Project Settings

Tailoring the Actions Profile

Open the ISPF/DB2 Actions Profile - Settings notebook. Use the Add push button to add the PL/I Compiler, Editor, Linker, and Make utility. For each, enter the values shown in Table 3 onto the pages of the notebook; then use the Add push button.

Table 3. Tailoring the Actions Profile

Action (Class)	General Page	Type Page	Options Page
COMPILE	name: PL/I Compile command: pli.exe	source: *.pli target: *.obj *.lst	DLL: ibmwf.dll entry: getcompileopts
EDIT	name: EPM Editor command: epm.exe	source: *.* target: (blank)	DLL: dde3def2.dll entry: edit
LINK	name: Link command: link386.exe	source: *.obj *.def target: *.exe *.map	DLL: dde3def2.dll entry: link
MAKE	name: Make Utility command: nmake.exe	source: *.mak target: (blank)	DLL: dde3def2.dll entry: make

Note: Use the Find push button to locate the DLLs. The IBMWF.DLL for the PL/I compiler is in the PL/I DLL directory (d:\IBMPLI\DLL), not in the WorkFrame/2 directory as written in the manual.

Figure 64 shows the final Actions Profile - Settings notebook.

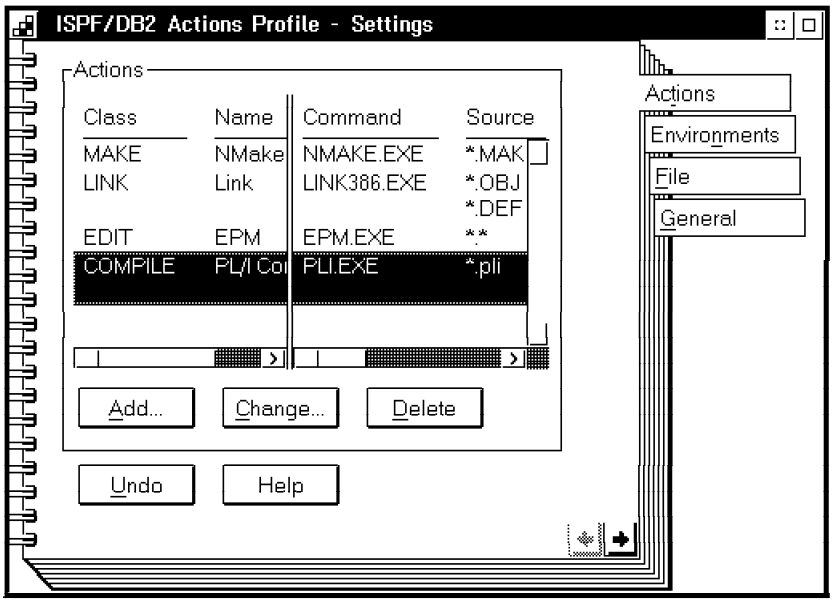


Figure 64. Tailored Actions Profile

Tailoring the Actions Profile Using Drag and Drop

You can add actions to the Actions Profile by copying from the Default Actions Profile. Open the Actions Profiles icon in the WorkFrame/2 folder, then open the Default Actions Profile.

Select an action in the list, drag and drop it to the actions list of the Actions Profile - Settings notebook of your project, while holding down the Ctrl key (copy). Use the Change push button to verify the parameters.

Compile Options

The WorkFrame/2 Actions Profile is appropriate if all programs have the same compile options. If they are different, then it is advantageous to put the additional options into the source file itself.

In our sample project we needed to run the SCLS102 program through the SQL preprocessor; therefore we added the preprocessor options as a first line:

```
*PROCESS PP ( SQL('dbname(sample) bind') );
```

Compile options for all source programs are set in the project notebook on the Actions page. Select the entry COMPILE::PL/I Compile and the Options push button and add compile options, for example, source, as shown in Figure 65.

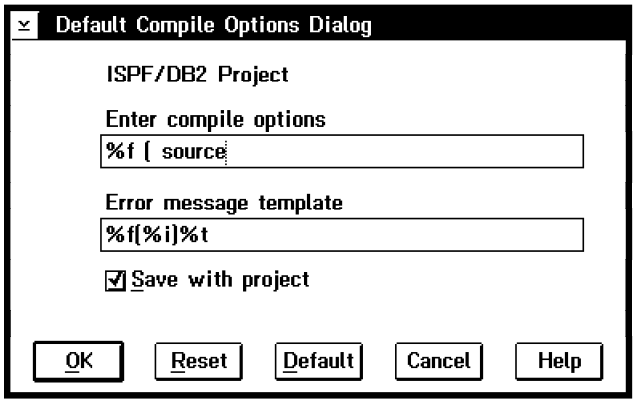


Figure 65. WorkFrame/2 Compile Options

Note: To understand the substitution variables (such as "%f") passed to the compiler use the Help facility:

- Click on the Help push button.
- Use Search under Services, enter %f, and start the search.
- Read up on "Substitution Variables" in the resulting window.

Link Options

On the Actions page of the Project - Settings notebook select the entry LINK::Link and the Options push button to add link options. For our application we selected the generation of a map file (SCLS101.MAP), and we specified the DB2 library, sql_dyn, in the Library list under File names (see Figure 66 on page 84).

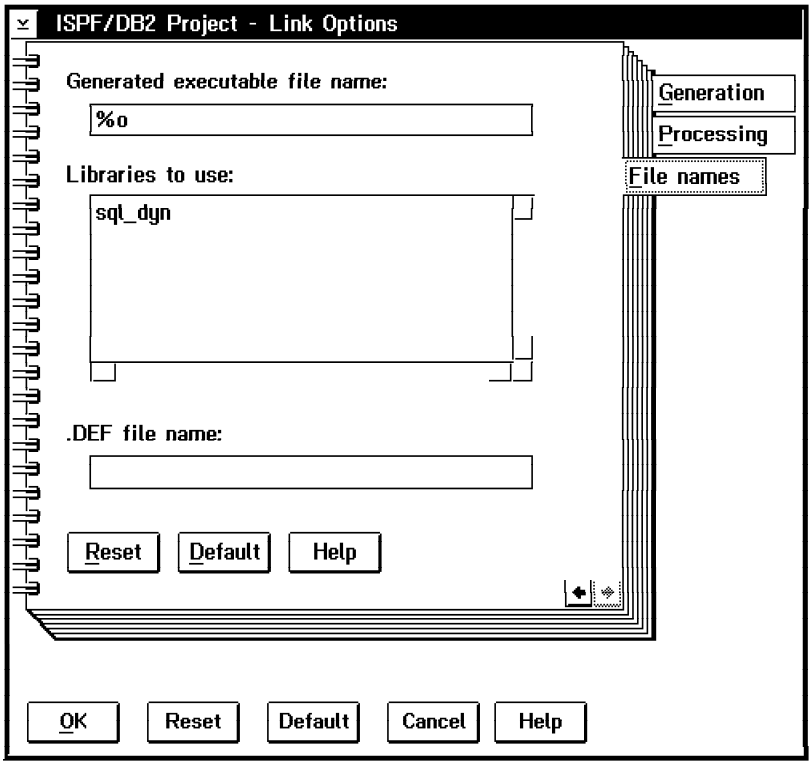


Figure 66. WorkFrame/2 Link Options

Compile, Link, and Run

Open the project (double-click) and use the pop-up menu for the icon of the SCLS101.PLI program. You can select the EDIT and COMPILE actions directly from the pop-up menu. Compile the SCLS102.PLI program in the same way; because of the PP(SQL) compile option, the SQL preprocessor is invoked as well.

Refresh the folder. Select the two object modules, SCLS101.OBJ and SCLS102.OBJ, and use the pop-up menu (for one of the two icons) to invoke the LINK action, which creates the executable, SCLS101.EXE.

Using the pop-up menu for the icon of the executable, invoke the RUN action to run the application.

Figure 67 shows the open project folder with source programs (.PLI), object modules (.OBJ), compile listings (.LST), link-edit map (.map), DB2 bind file (.BND), executable (.exe), and make file (.mak, .Dep).

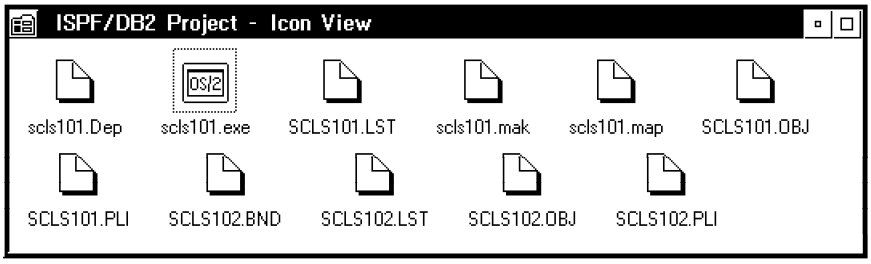


Figure 67. ISPF/DB2 Project Folder

Using a Make File

For larger projects it is important to automate compile and link actions. A make file contains a list of actions to be performed on the source programs of a project, for example, compile and link. It also contains a list of dependencies, so that programs are compiled only if some source on which they depend has changed. The NMake utility runs through all of the actions in a make file.

Creating a make file can be a difficult task when done manually. WorkFrame/2 provides the MakeMake utility to create a make file for the project.

Note: Visual PL/I builds the make file and runs it as well.

Creating the Make File

To start the utility, use the pop-up menu for the project icon, select Actions and then MAKEMAKE in the submenu.

Note: You can find the MakeMake utility in the Tools folder from the WorkFrame/2 folder. You can also drag and drop a project onto the MakeMake icon to start the utility.

In the MAKEMAKE dialog, select the compile and link actions as shown in Figure 68.

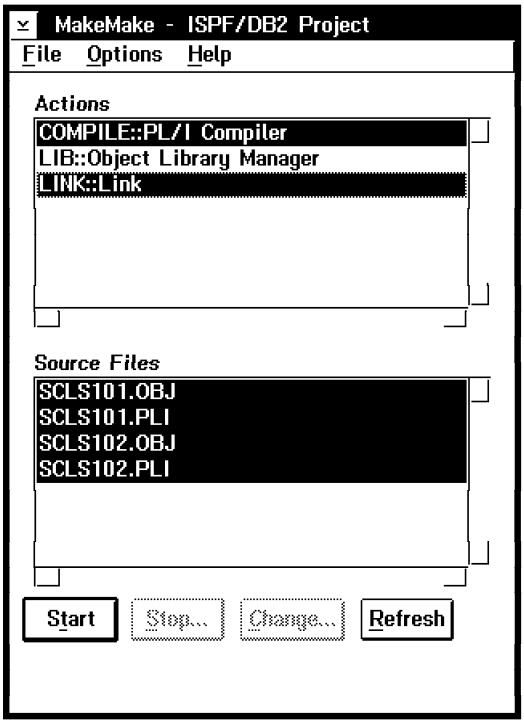


Figure 68. MakeMake Dialog

When you select the action, the list of source files is automatically created. Now use the Start push button to run the utility.

After a few moments the MakeMake Results window is displayed as shown in Figure 69 on page 86.

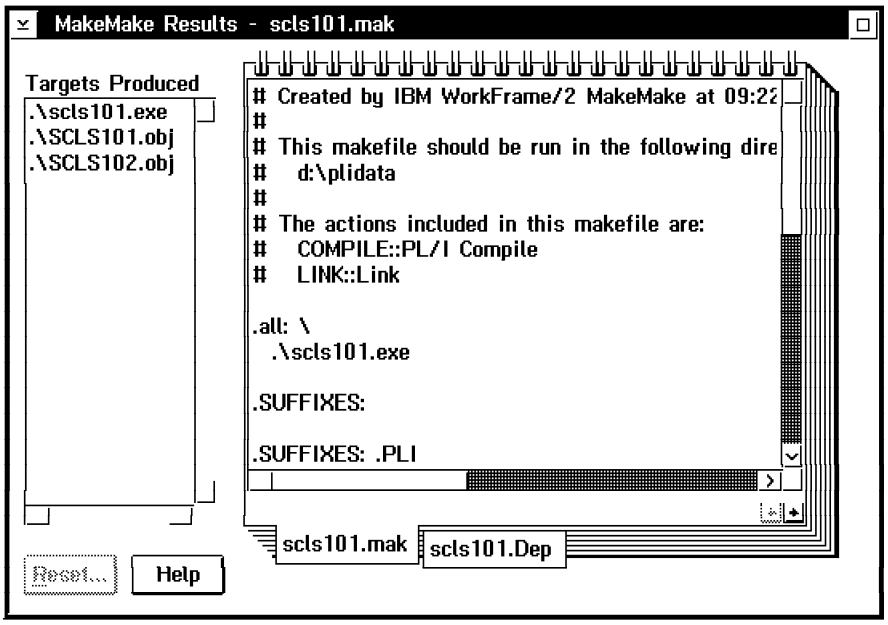


Figure 69. MakeMake Results

Close the Results window, and save the make file using the File pull-down in the MakeMake dialog.

Running the NMake Utility

Refresh the project folder and use the pop-up menu for the make file icon itself (SCLS101.mak in our example). Select the MAKE action to run the utility.

A window is opened and you can monitor all of the compilations and link-edit actions. The process runs until an error occurs, or to completion, as shown in Figure 70.

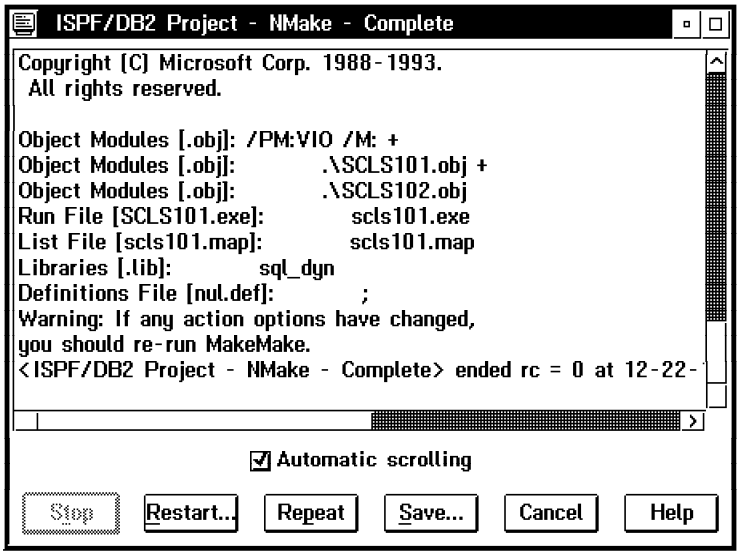


Figure 70. Running the NMake Utility

Automatic Scrolling: The monitor window of the NMake utility scrolls automatically with the last action if the Automatic scrolling box is checked. To make scrolling the default, use the setup facility of WorkFrame/2.

Open the Setup icon in the WorkFrame/2 folder (see Figure 60 on page 79). Open the Settings under Options and check the box as shown in Figure 71. We also checked the Refresh box for the project so that the folder is always current.

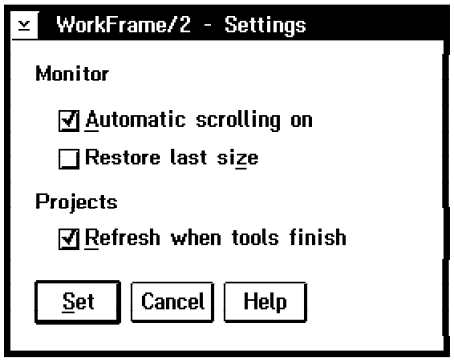


Figure 71. WorkFrame/2 Settings

Action Log

All actions are logged in the Action Log. Open the Action Log from your project folder. Figure 72 shows a sample Action Log with edit, compile, link, MakeMake, and NMake.

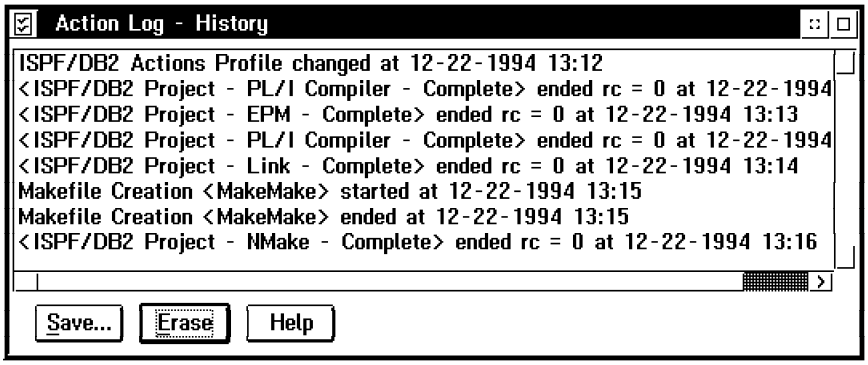


Figure 72. Action Log

Integration with the Editor

If your compilation fails, you will see error messages in the monitor window of the PL/I compiler. You can double-click on the error message to invoke the editor (EPM) for the source file.

We changed one line in SCLS101 from “display” to “displax,” simulating a typing error. The monitor window shows the error messages (see Figure 73 on page 88).

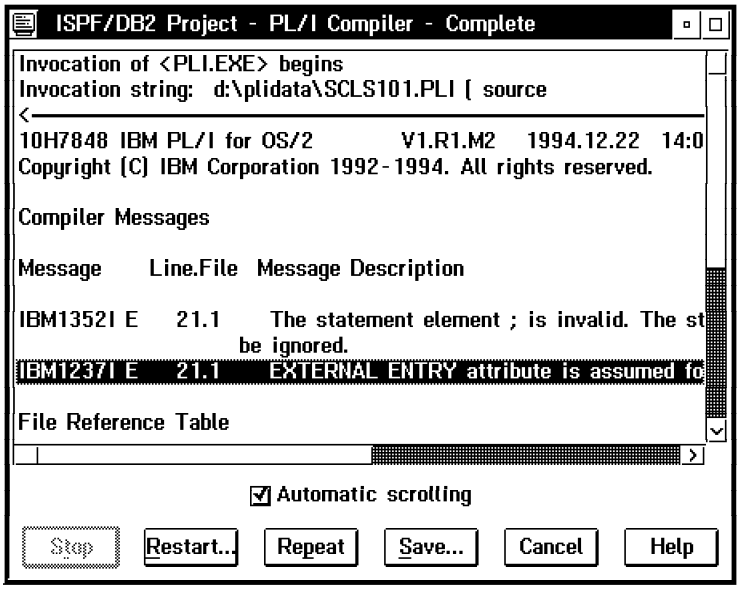


Figure 73. Monitor with Compilation Error

Double-click on one of the error messages (highlighted in Figure 73), and the EPM editor appears with the source program.

You may even position the editor at the line in error. The fastest way to achieve this is to have an empty EPM window already started. Just drag and drop the error message into the EPM editor window. EPM brings up the source, highlights the line (in red), and positions the cursor as shown in Figure 74.

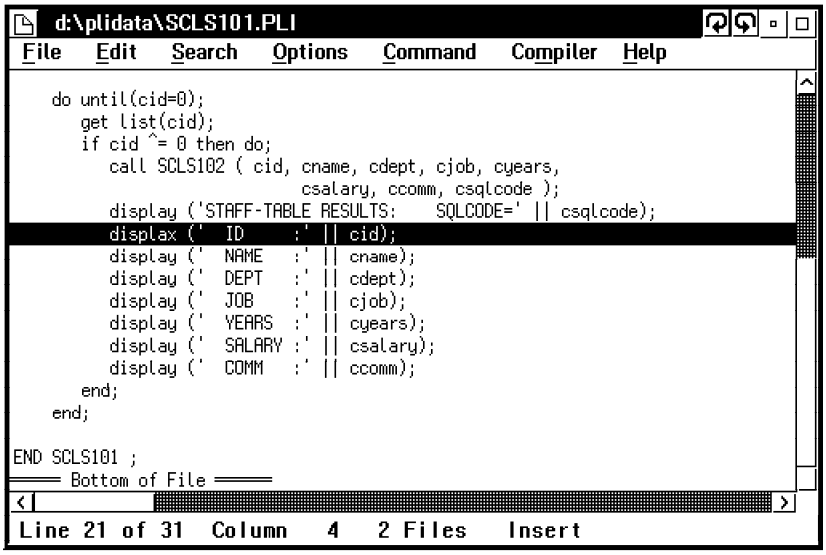


Figure 74. Editor with Line in Error

Adding Own Actions

If it were necessary to rebind the DB2 plan for your application, you would use an SQLBIND command:

```
SQLBIND scsl1O2.bnd SAMPLE /I=CS
```

Instead of issuing such a command in an OS/2 window, however, you can define SQLBIND as a WorkFrame/2 action.

Open the Actions Profile of the project. Use the Add push button to define a new action. We defined DB2 Bind in the BIND class and ran it in an OS/2 window (see Figure 75).

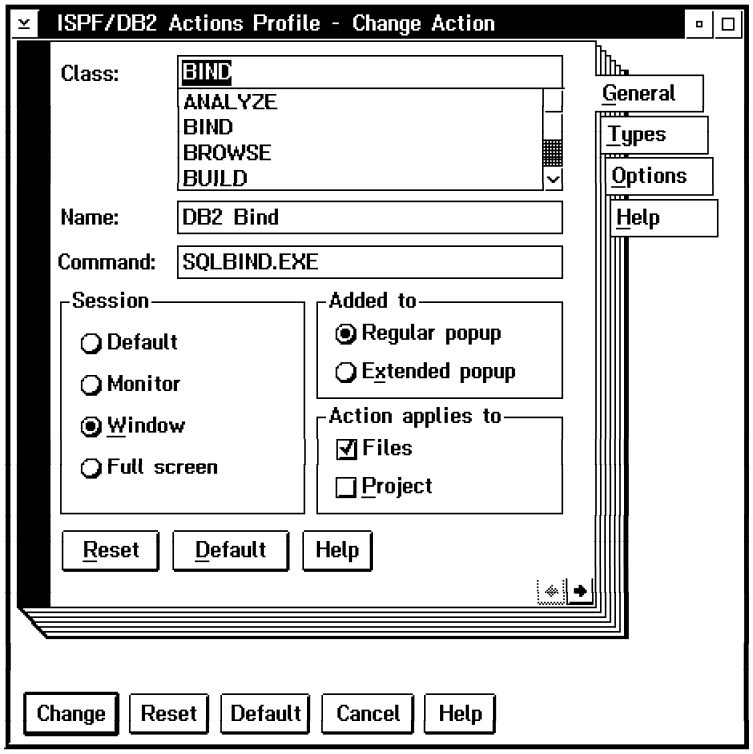


Figure 75. Defining DB2 Bind As an Action

For types, we defined *.BND, and in the Options we selected the DDE3DEF2.DLL with entry point DEFAULT.

To pass proper parameters to DB2 Bind we opened the project settings and under Actions we defined the Options for DB2 Bind as shown in Figure 76 on page 90.

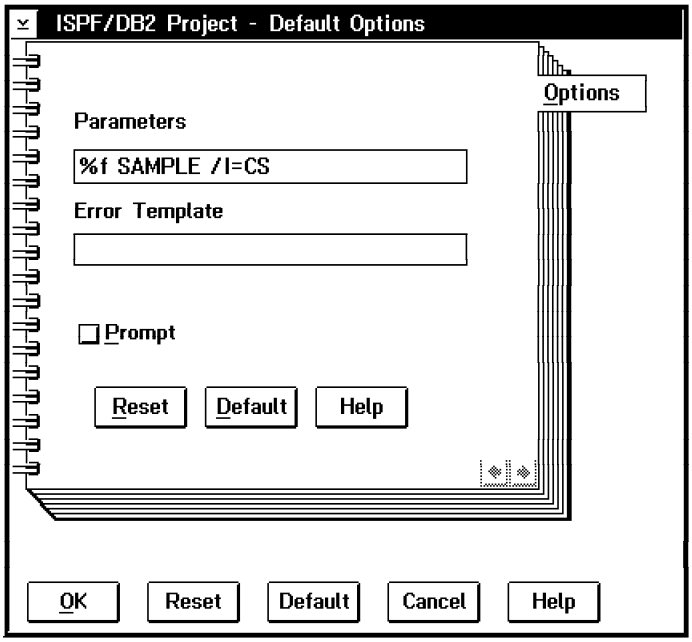


Figure 76. Defining DB2 Bind Parameters

We then used the pop-up menu for the bind file (SCLS102.BND) and invoked BIND to rebind the DB2 plan.

Similarly you can add any new tool to WorkFrame/2. Instead of adding the new action to an individual project, you can add it to the Default Actions Profile (in the WorkFrame/2 folder) and then copy the action from there to any project where it is needed.

Integrating WorkFrame/2 with Visual PL/I

Many times your PL/I application contains visual and nonvisual components. You can manage the nonvisual components with WorkFrame/2 as discussed in this chapter, and the visual components with Visual PL/I as described in Chapter 5, “Visual PL/I of the PL/I for OS/2 Toolkit” on page 35.

Visual PL/I also manages applications using make files. It creates the make file as part of generation and then runs it immediately.

If there is a need to link together nonvisual object modules created using WorkFrame/2 (or compiled using the command interface) with Visual PL/I object modules, you must be very careful.

You can enter the names of the nonvisual object modules in the Visual PL/I Project Settings (see Figure 50 on page 67) to link them to the visual object modules.

However, this dependency is not recorded in the make file that Visual PL/I generates. If the nonvisual module changes, rerunning the make file of the Visual PL/I application will not relink the program. Erase the executable and rerun the make file.

Note: You could change the make file and record the dependency of the executable on the nonvisual object module. However, by default, Visual PL/I re-creates the make file every time you generate the application. Open the Visual PL/I Project Settings and change the Output Options to indicate that the make file (.mak) should not be generated.

Chapter 7. CODE/370 for Host Testing

In this chapter we explore the use of CODE/370 for working with host PL/I application programs.

If you want to fully understand the concepts of CODE/370, you should read the *Cooperative Development of High-Level Language Applications Using CODE/370 and LE/370*, GG24-3872. Most of the content of that redbook still applies for release 2 of CODE/370. The user interface on the workstation is different because release 2 uses a new editor (LPEX), but the concepts are basically the same.

Another good source of information is the *CODE/370 Self-Study Guide*, SC09-2047, which guides you through the operations of CODE/370 by example.

In this chapter we concentrate on using CODE/370 for PL/I programs.

Sample Programs

To work with CODE/370 we used some of the sample programs described in previous chapters:

SCLS101 ISPF main program with panel displays

SCLS102 ISPF subroutine to retrieve data from DB2

PCASH CICS DB2 program to retrieve cashier information

PNEWOR3 CICS DB2 program to update order information

PLISTUB CICS transaction program to invoke PCASH and PNEWOR3.

Installation and Tailoring

The CODE/370 product is shipped as a host product. Each user who wants to use the workstation part of the product must install (download) the product from the host.

The installation procedure is described in detail in *CoOperative Development Environment/370 Installation*, SC09-1624.

The workstation product contains three components:

- CODE workstation component
- LPEX editor component
- WorkFrame/2 component.

The host product data sets are by default called EQAW.V1R2M0.SEQAxxxx. If your naming convention is different, have your administrator tailor the package members of the

workstation product in the SEQAIENU partitioned data set (PDS). In this data set, members EVFILPXI, EVFIMVSI, and EVFIWFI refer to the host PDS in the HOSTLOC keyword. Change that value, EQAW.V1R2M0, to your actual host data set prefix.

In abbreviated form, the installation procedure includes the following steps:

1. Create a directory for the installation utility, for example, CODEUTIL, and make it the current directory.
2. Download the installation command:

   ```
   RECEIVE CODEINST.CMD s:'EQAW.V1R2M0.SEQAOS2(CODEINST)'
   ```

   Your TSO host session letter is `s` in the above command.
3. Start the installation utility:

   ```
   CODEINST /S s /O MVS /D dataset.prefix
   ```

   All three components are installed in sequence. You can choose directory names and have the CONFIG.SYS file updated automatically.
4. Shut down and restart your OS/2 system to make all changes effective.

After installation you will find a new folder, CODE/370, on your desktop (see Figure 77).

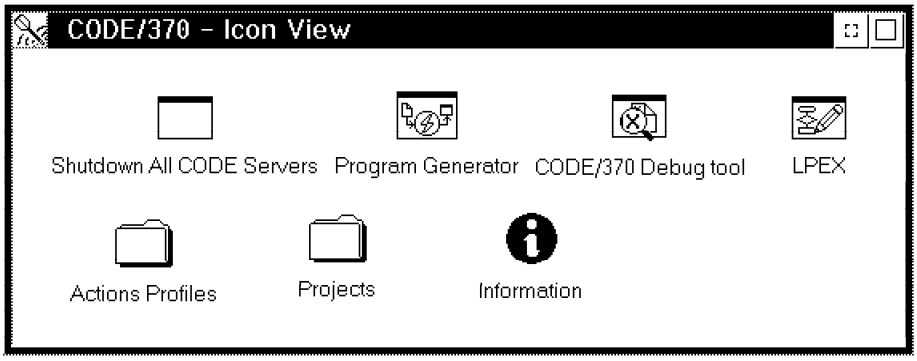


Figure 77. CODE/370 Folder

Configuring Communications

The workstation component of CODE/370 communicates with the host component through either APPC LU 6.2 or SRPI LU2.

The setup of the communications is described in detail in the installation manual. The APPC setup is more complex than the SRPI setup, but it performs better, and only APPC allows the debugging of batch applications, an important new feature of release 2.

To keep things simple, we only used the SRPI protocol when testing examples for this document. The user interface is the same for both protocols.

Preparing the Host Data Set for SRPI

Each user who wants to use CODE/370 must have a data set called userid.CODEPROD.EVFMPARM, which contains one line of 80 bytes with:

```
EVFMINIT   TYPE(I)
```

Before you can use CODE/370 on the workstation, you must start the CODE/370 server using the CLIST EVFSTART. The TSO session should display the MVSSERV logo.

Host Naming Convention

The examples described in this chapter assume that host data sets are named according to a strict naming convention. All data sets used for CODE/370 use a prefix of userid.CODE:

```
userid.CODE.PLI        - PL/I source
userid.CODE.OBJ        - Object code
userid.CODE.PLISTNG    - Compile listing
userid.CODE.LOAD       - Load modules
userid.CODE.LKEDLIST   - Linkage editor listing
userid.CODE.DB2PDSRC   - Output source of DB2 preprocessor
userid.CODE.DB2PCLST   - Listing of DB2 preprocessor
userid.CODE.CICSPLI    - Output source of CICS preprocessor
userid.CODE.CICSLIST   - Listing of CICS preprocessor
userid.CODE.xxxx       - others
userid.DBRMLIB.DATA    - SQL DBRM members
```

Using the LPEX Editor

To work with CODE/370 on the workstation, start the LPEX editor (see Figure 78).

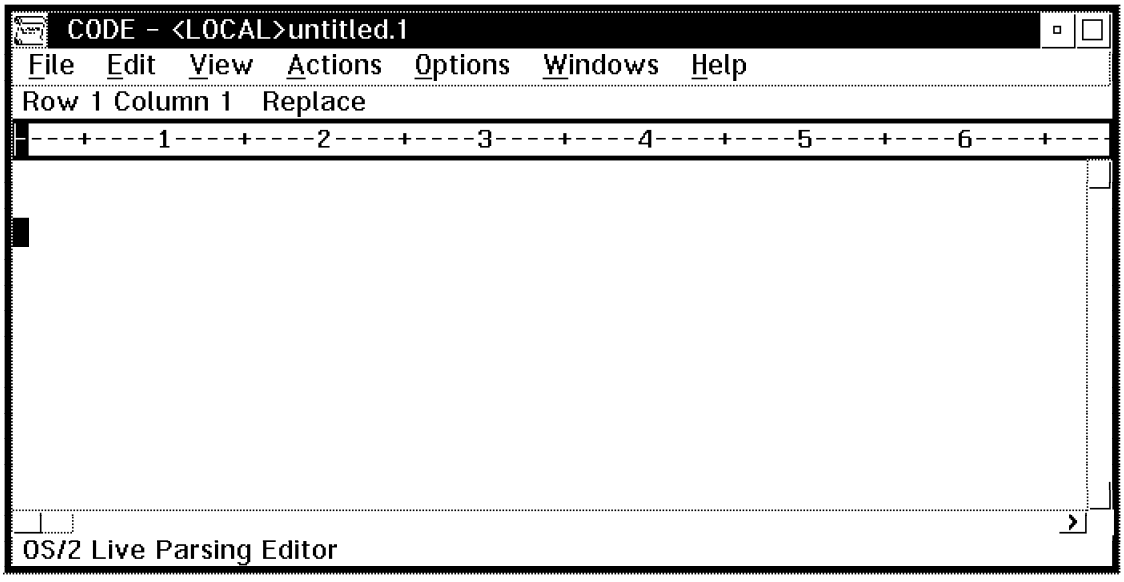


Figure 78. LPEX Editor

Opening a Host Program for Editing

To download a PL/I program for editing with LPEX, open the file dialog as follows:

1. From the File pull-down in the LPEX editor main window, select Open for edit... to get the Select file - Open for Edit window.
2. Select the SRPI server in the Containers/Directories list that matches your TSO session.
3. In the file name field enter userid.CODE.*.
4. Click on the Expand push button. The list of data sets is shown in the window on the left.
5. Select the source data set, userid.CODE.PLI, and click on the Add push button. The member list is displayed in the window on the right.
6. Select the member (PCASH) and click on OK.

Figure 79 shows a snapshot of the Select file - Open for Edit dialog.

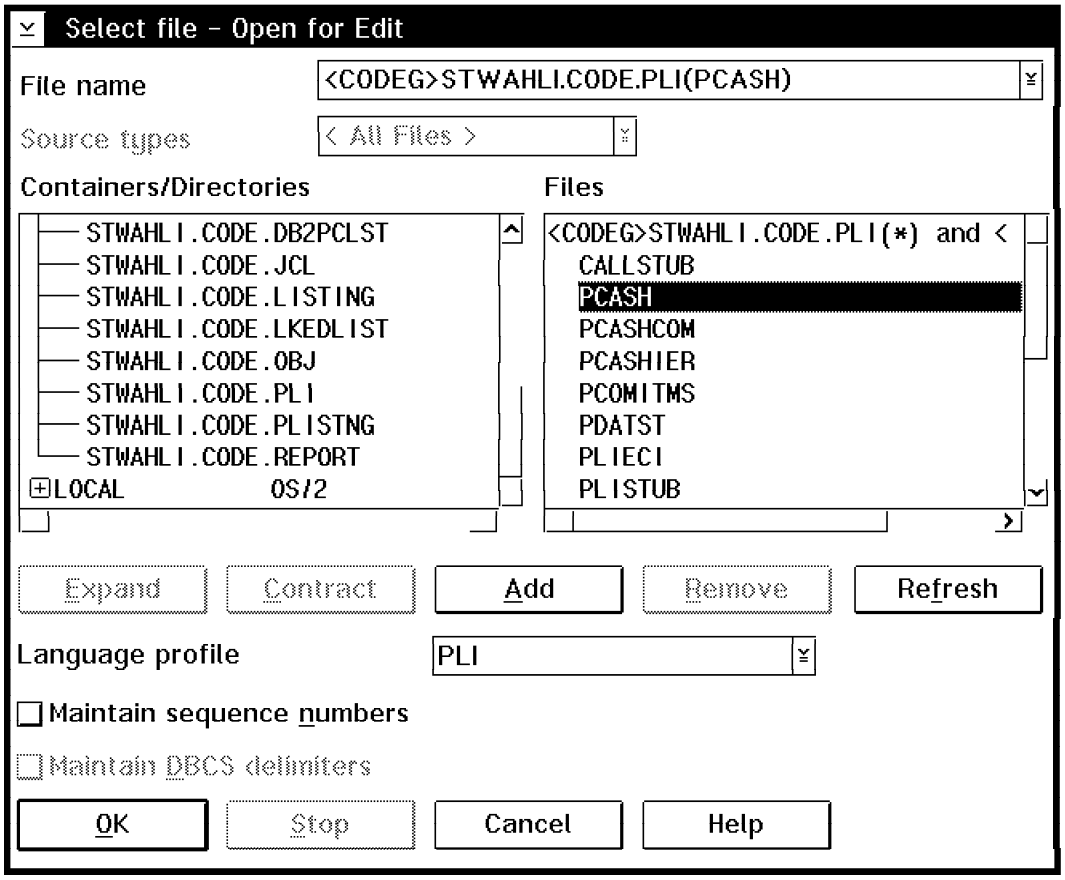


Figure 79. LPEX Editor: Downloading a PL/I Source Member

LPEX is a language-sensitive editor. Different kinds of source statements and keywords are displayed in different colors. Figure 80 on page 95 shows an extract of the source program as displayed by LPEX. Note that the different colors used for keywords (PROC, END, EXEC SQL), strings ('PCASHIER'), and comments (/* xxx */) do not show in this black and white snapshot.

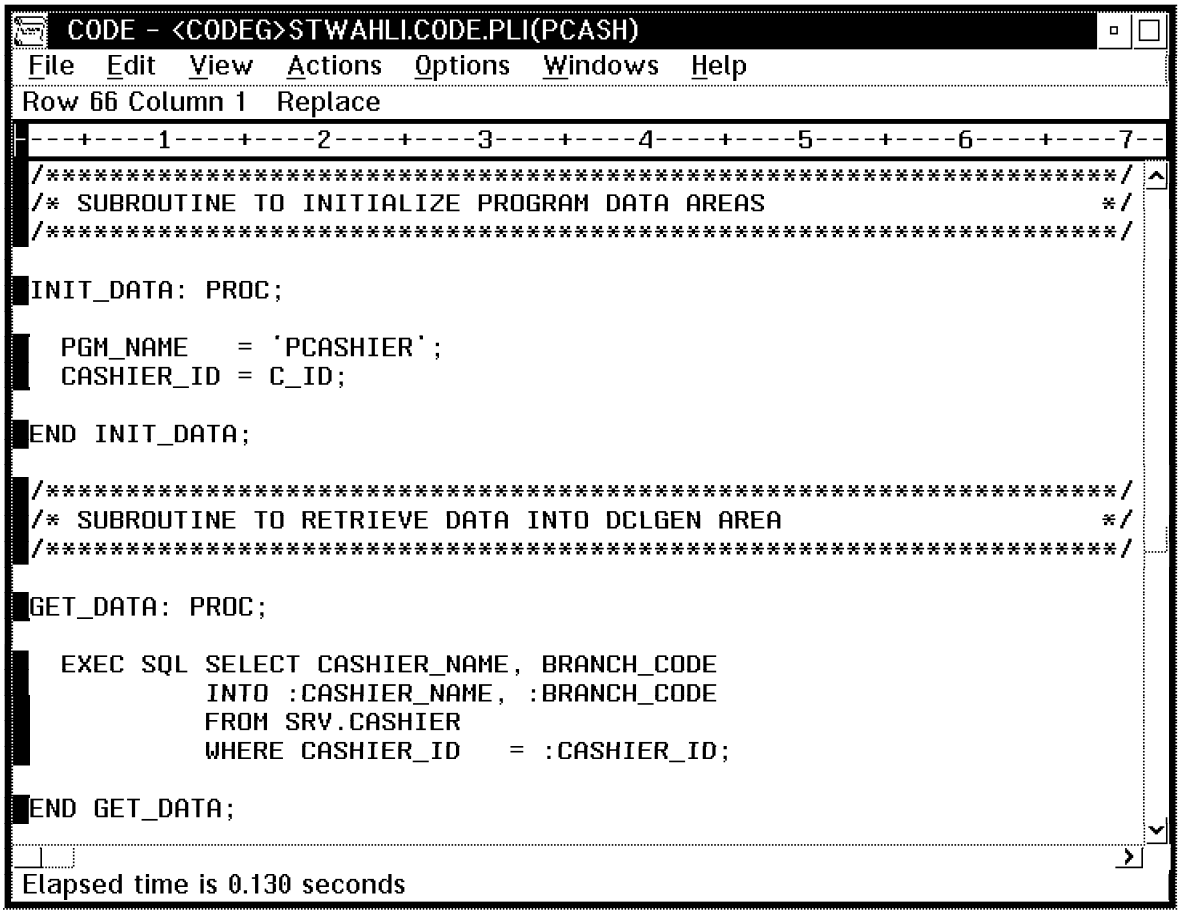


Figure 80. LPEX Editor: PL/I Source Program

Editing with LPEX

You can open multiple editor windows using the File pull-down. You can edit programs stored on the workstation by expanding the LOCAL OS/2 line in the container list and selecting directories on the hard disk. The same language-sensitive editing applies to all source programs.

LPEX has multiple language profiles and parses the source program according to the language.

Experiment with the View and Options pull-downs to explore the power of this new editor.

Compiling and Linking Host Programs

After making changes to the source program save it (on the host) using the Save option in the File pull-down of the LPEX editor window. Now you are ready to compile the program.

Compile and link activities are performed using CLIST or REXX procedures on the host. These procedures are invoked from the workstation.

Compile and Link Procedures

CODE/370 comes with many predefined compile and link procedures for different languages. These procedures are stored in the host data set, SEQACLIS. Make sure that this data set is in the SYSPROC concatenation. Examples of procedures are:

```
COBCOMF   - Compile a COBOL program in TSO foreground
CCOMF     - Compile a C program in TSO foreground
PLICOMF   - Compile a PL/I program in TSO foreground
PLICOMB   - Compile a PL/I program in batch (generate a job)
CICSCPF   - Compile a PL/I CICS program in TSO foreground
LKEDF     - Link-edit programs in TSO foreground
etc.
```

Procedures either run in foreground or submit a job for batch processing.

Note: If your CODE/370 data sets have names different from the default (EQAW.V1R2M0.xxxx), you must tailor the procedures to match your naming convention.

The PL/I compiler ships the following CLISTs, which you can invoke from a CODE session:

```
IELFCOMP  - Compile a PLI program in TSO foreground
IELBCOMP  - Compile a PLI program in batch (job)
IELFLINK  - Link-edit a PLI program in TSO foreground
IELBLINK  - Link-edit a PLI program in batch (job)
```

Preprocessors

Many PL/I programs will need to be processed by preprocessors and compilers. Typical preprocessors are the DB2 SQL and CICS preprocessors.

REXX procedures could be developed for each preprocessor separately, but it is easier to invoke just one procedure and have all preprocessors and the compiler invoked in one task.

We developed a single REXX procedure, PLICOMP, that can be used to run a PL/I source program through the SQL and/or CICS preprocessor and the PL/I compiler. A parameter is passed to the procedure to specify which preprocessors must be invoked before the compiler:

```
PP (SQL)             - invoke the SQL preprocessor
PP (CICS)            - invoke the CICS preprocessor
PP (SQL CICS)        - invoke the SQL and CICS preprocessor
PP (MACRO SQL CICS)  - invoke PL/I macro processor and SQL and CICS
```

See “Program PLICOMP” on page 143 for a listing of the procedure.

Link

In many cases multiple object programs are linked into a single load module. CODE/370 provides two REXX procedures, LINKF and LINKB, to link objects into a load module.

For DB2 and CICS programs additional modules must be linked to the load module. We developed a new REXX procedure, LELINKF, to link multiple object modules into a load module, together with appropriate modules for DB2 and CICS. Parameters are passed to the procedure to indicate which subsystems are used:

```
CICS    - link with CICS interface module DFHELII
SQL     - link with SQL interface module DSNELI (TSO) or DSNCLI (under CICS)
```

This procedure assumes that you have three simple members containing one line of 80 bytes stored in your object library:

```
member:     content:
-------     ------------------------
DFHELII      INCLUDE SYSLIB(DFHELII)
DSNELI       INCLUDE SYSLIB(DSNELI)
DSNCLI       INCLUDE SYSLIB(DSNCLI)
```

The SYSLIB concatenation is built so that the DB2 and CICS library data sets are included.

See “Program LELINKF” on page 146 for a listing of the procedure.

Bind

For programs containing SQL statements the DB2 BIND process must be run to create an application plan. Usually multiple database request modules (DBRMs) are bound into a single plan.

CODE/370 provides two REXX procedures, BINDF and BINDB, to bind DBRMs into a plan. The parameters for these procedures are:

```
BINDF (member,member,...) plan action isolation

plan      :  name of resulting plan      default: first member
action    :  ADD or REPLACE              default: REPLACE
isolation :  RR or CS                    default: CS (cursor stability)
```

See “Program BINDF” on page 149 for a listing of the procedure.

Invoking the Compile and Link Procedures

The compile and link procedures can be invoked from either the LPEX editor, the command log, or the program generator. The program generator is started from the CODE/370 folder (see Figure 77 on page 92) or through the Compile selection in the Actions pull-down. Note that the Compile selection is available only if the language is set to PL/I MVS.

Invoking a Compile from the Command Log

Select the Command Log from the Windows pull-down of the LPEX editor. Type the command in the box at the bottom of the window and click on the Process push button. The command is executed at the host, and the output is returned to the upper portion of the window when the command has finished. See Figure 81 on page 98.

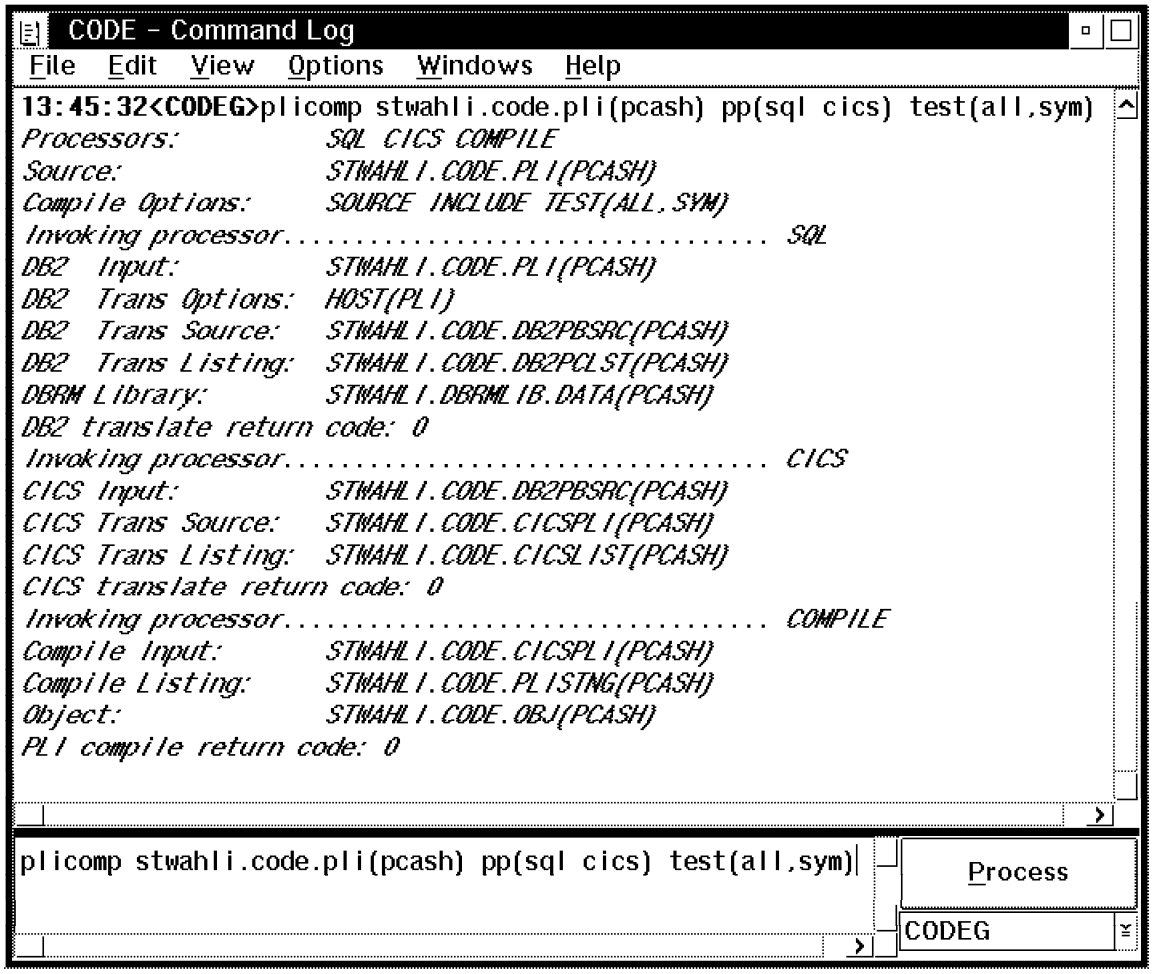


Figure 81. Command Log: Compiling a Program

Invoking a Compile from the Program Generator

The program generator provides facilities to select the source file, userid.CODE.PLI(member), the command procedure on the host (PLICOMP), and compile options.

Figure 82 on page 99 shows the program generator window with the selected source file and the command procedure, PLICOMP.

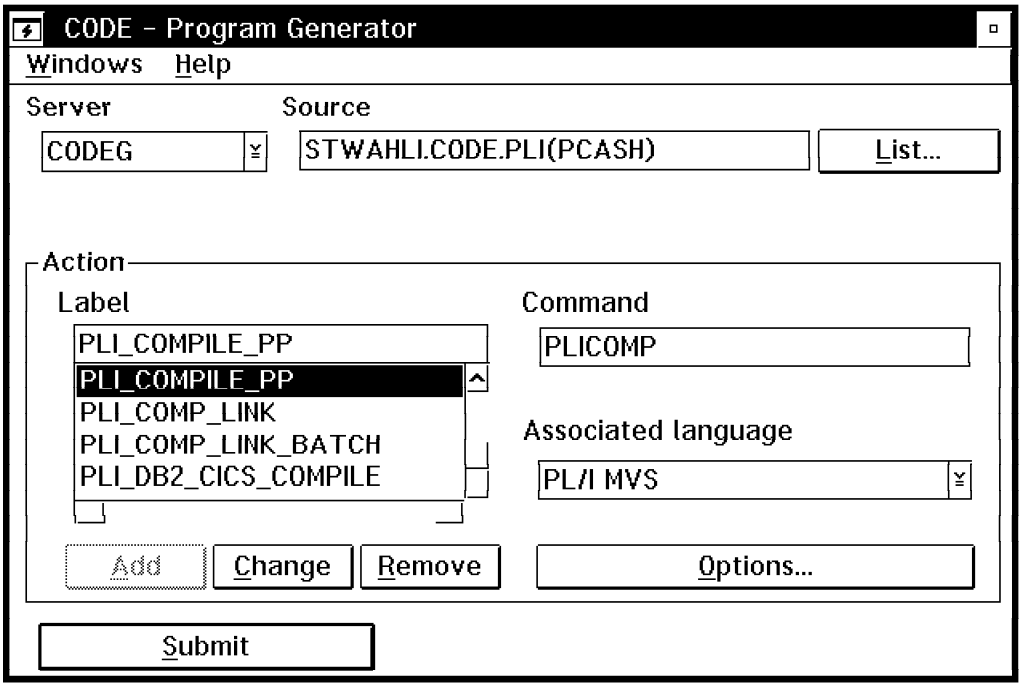


Figure 82. Program Generator: Submit Program Compilation

Adding Actions to the Program Generator

We previously added the PLICOMP command procedure to the list of actions using the Add push button.

In a similar way we added our new procedures, LELINKF and BINDF, to the list of actions. For those procedures the Associated language is set to User defined/MVS.

Compile Options

Before submitting the command use the Options... push button to specify the PL/I compile options. See Figure 83 on page 100 for an example of selecting the TEST compile option using the Compile Options Dialog.

Note: The EVENTS option is not supported by the current PL/I compiler and no SYSEVENTS file is created for downloading to the editor. The EVENTS option is supported by the new COBOL and C compilers.

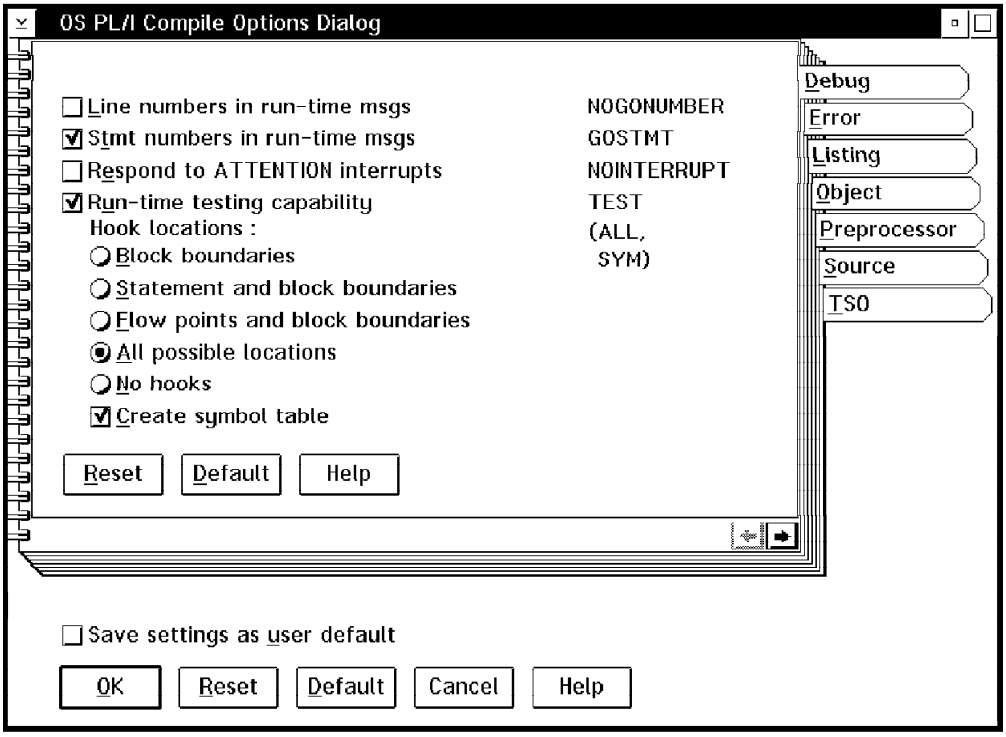


Figure 83. Program Generator: Compile Options

The Compile Options Dialog does not allow specification of preprocessor options. To specify preprocessor options you can use the Associated language User defined/MVS and set preprocessor and compile options through the Options... push button. See Figure 84 for an example of User Command Options.

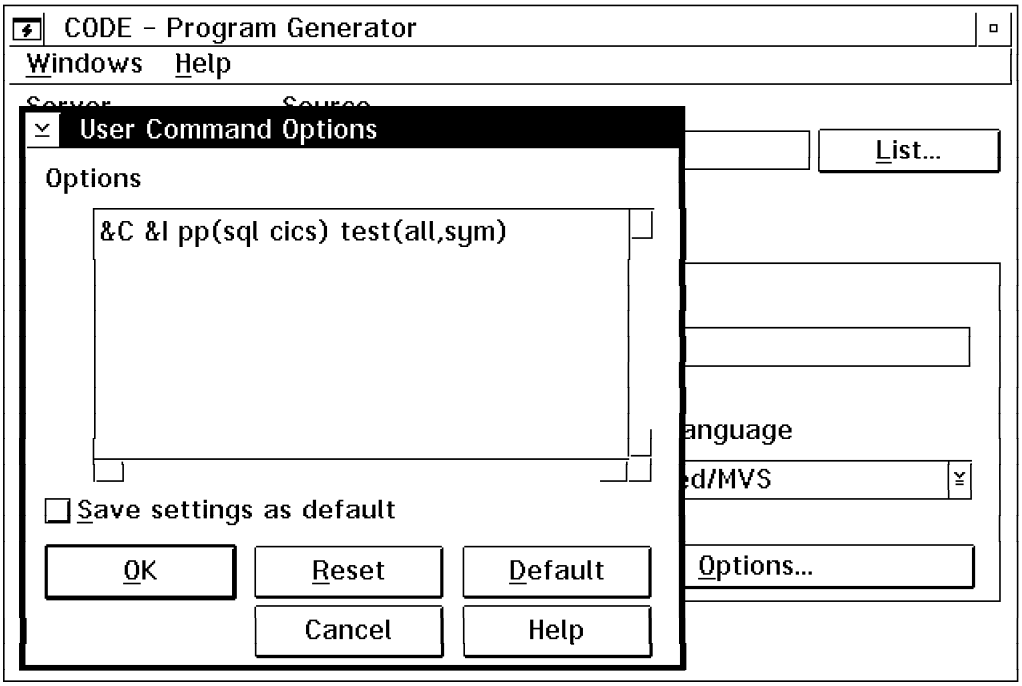


Figure 84. Program Generator: Preprocessor and Compile Options

Note: The two ways of passing compile options to the host command procedure are mutually exclusive. You can either specify compile options through the notebook (Associated language PL/I MVS, Figure 82 on page 99) or as a string (Associated language User defined/MVS, Figure 84).

Preparing the Sample Programs

The sample programs listed in "Sample Programs" on page 91 were compiled using the PLICOMP procedure:

```
PLICOMP userid.CODE.PLI(SCLS101)              GOSTMT
PLICOMP userid.CODE.PLI(SCLS102) PP(SQL)      ATTRIBUTES
PLICOMP userid.CODE.PLI(PCASH)   PP(SQL CICS) TEST(ALL,SYM)
PLICOMP userid.CODE.PLI(PNEOR3)  PP(SQL CICS) SYSTEM(CICS)
PLICOMP userid.CODE.PLI(PLISTUB) PP(CICS)
```

The programs were link-edited using the LELINKF procedure:

```
LELINKF userid.CODE.OBJ(SCLS101,SCLS102) SQL
LELINKF userid.CODE.OBJ(PCASH)           CICS SQL
LELINKF userid.CODE.OBJ(PNEWOR3)         CICS SQL
LELINKF userid.CODE.OBJ(PLISTUB)         CICS
```

The DB2 programs generate DBRMs, which must run through DB2 bind:

```
BINDF (SCLS102)
BINDF (PCASH,PNEWOR3) PLISTUB
```

Host Setup to Run the Sample Programs

The ISPF sample program (SCLS10x) requires several panels in the ISPPLIB concatenation. A CLIST is used to set proper execution options and invoke the program under DB2. The basic commands are:

```
DSN SYSTEM(DSN)
  RUN PROGRAM(SCLS101) PLAN(SCLS102) PARMS('execution-options/')
  END
```

The CICS programs require that a transaction code (TCLL) be defined in CICS to invoke the main program, PLISTUB. In addition, the resource control table (RCT) must define that the TCLL transaction use the matching DB2 plan, PLISTUB. The programs are invoked in a CICS terminal window through:

```
TCLL PCASH      - run PLISTUB and invoke PCASH
TCLL PNEWOR3    - run PLISTUB and invoke PNEWOR3
```

The CODE Debug Tool

The CODE Debug Tool is a powerful tool to monitor your PL/I programs during execution. It comes in two versions, a mainframe interactive (MFI) debug tool, and a cooperative debug tool with a workstation user interface.

The Debug Tool replaces the PLITEST facility of previous host PL/I compilers.

TEST Compile Option

To use the Debug Tool, you must compile the programs with the TEST compile option. To get the most from the Debug Tool use the option:

```
TEST(ALL,SYM)
```

This option generates all possible hooks into the object code and allows for the most detailed debugging of your code.

TEST Execution Time Option

To start the Debug Tool at execution time you can use the TEST execution time option of Language Environment/370 (LE/370). Depending on your execution environment you can pass the execution options as parameters or add them by link-editing a user options module, CEEUOPT.

For TSO and DB2 programs execution options can be passed as parameters:

```
TSO:  CALL 'userid.CODE.LOAD(member)'  'execution-options/program-parms'
DB2:  RUN PROGRAM(name) PLAN(plan) PARMS('execution-options/program-parms')
```

The slash is necessary to separate execution time options from program parameters; it must even be present if no program parameters are passed:

```
TSO:  CALL 'userid.CODE.LOAD(member)'  'TEST(ALL,*,;,)/'
```

The execution time option to invoke the Debug Tool is:

```
TEST(ALL,*,;,MFI:)     - MFI Debug Tool (default)
TEST(ALL,*,;,LU2:)     - PWS Debug Tool using LU2 protocol
```

Execution Time Options Module

For programs executing under CICS or IMS, the execution options must be compiled into the user options module, CEEUOPT, and then link-edited to the load module.

LE/370 provides an assembler macro, CEEXOPT, to compile execution options into an options module. You can use the REXX procedure, ASMCOMF, of CODE/370 to compile this module into object code. Link-edit the CEEUOPT object code with your programs:

```
LELINKF userid.CODE.OBJ(PCASH,CEEUOPT)    CICS SQL
LELINKF userid.CODE.OBJ(PNEWOR3,CEEUOPT)  CICS SQL
LELINKF userid.CODE.OBJ(PLISTUB,CEEUOPT)  CICS
```

Debug Tool Data Sets

Before invoking the Debug Tool you can allocate some data sets to control its initialization:

INSPPREF Preferences file, contains initial commands to be executed.

INSPLOG Log file of all commands executed during the session.

INSPSAFE File to save options.

To use the preferences file, pass its name in the execution time options:

```
TEST(ALL,*,;,MFI:INSPPREF)         - MFI debug tool
TEST(ALL,*,;,LU2:d:\code\inspprefc) - PWS debug tool
```

For the MFI Debug Tool you specify a ddname; for the PWS Debug Tool you specify an actual file on your workstation.

Preferences File

The preferences file can be used to specify the location of the source listings of your programs. The debugger needs to know from where to load the source so that the execution of the program can be viewed in a debug window.

The default file name of the source listing is userid.membername.LISTING. In our example we stored the listings in a PDS, userid.CODE.PLISTNG. The commands in the preferences file set the proper location:

```
SET SOURCE ON (PCASH)   userid.CODE.PLISTNG(PCASH);
SET SOURCE ON (PNEWOR3) userid.CODE.PLISTNG(PNEWOR3);
```

Source Listing

The source listing shown by the Debug Tool is the output listing of the compiler. If you have run your original source through an SQL or CICS preprocessor, you will see all of the expansions of those preprocessors.

Note: The PLITEST facility of PL/I for OS/2 shows the original source (see Chapter 4, “PL/I for OS/2 Test Facility” on page 31).

MFI Debug Tool

The MFI Debug Tool works in TSO foreground. By default the screen is divided into three areas:

- The monitor area, which shows values of variables and expressions during the execution of the program. To monitor the value of the SQLCODE variable enter MONITOR LOCAL LIST SQLCODE.
- The source area, which contains the program listing. It is used to show the animated execution of the program, highlighting the current line, and to set and remove breakpoints.
- The log area, which shows all commands that have been executed and their results. For example, LIST SQLCODE displays the value of that variable in the log area.

Figure 85 on page 104 shows the initial display when program SCLS101 is started using the TEST execution time option.

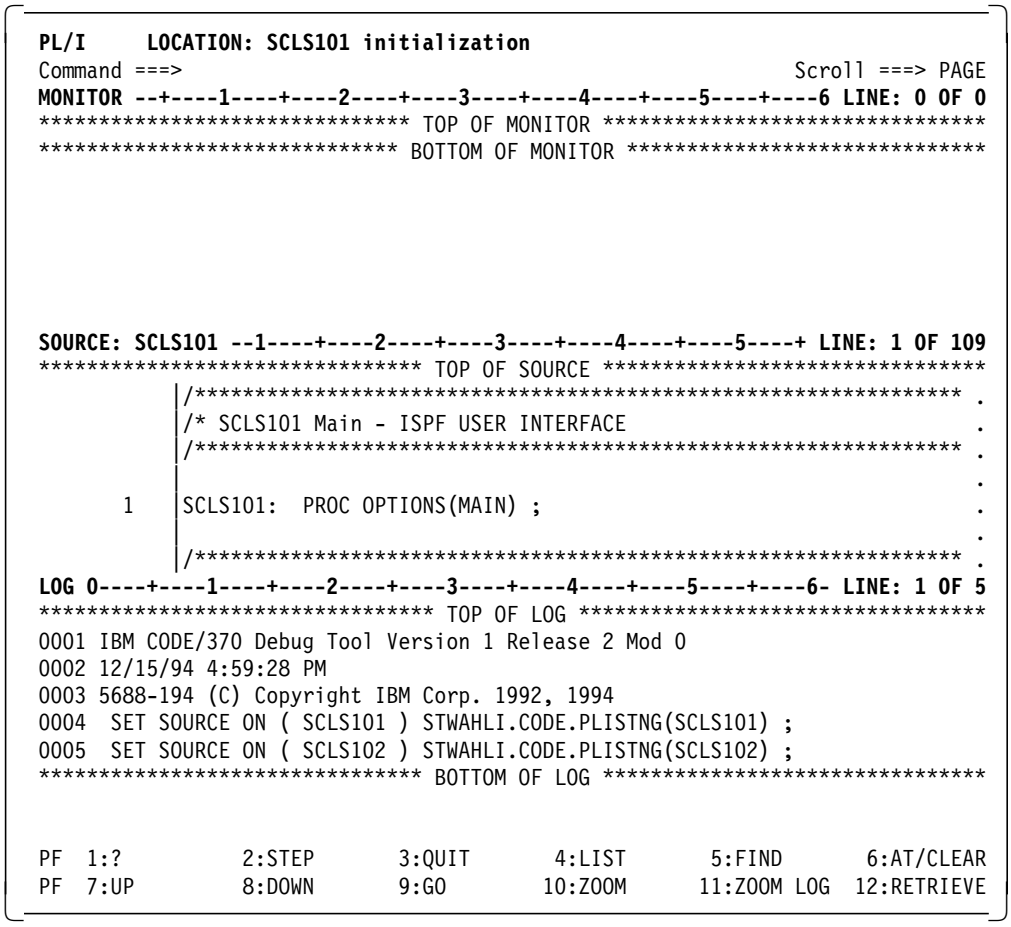


Figure 85. MFI Debug Tool: Initial Screen

Note: Different colors are used to highlight the areas of the screen.

You can now step through the program using the STEP command (or PF2). When a call for another program is issued, the new program listing is loaded and execution continues.

Figure 86 on page 105 shows a snapshot after the SQL SELECT statement in program SCLS102 has been executed. Several variables have been added to the monitor area. The commands that have been executed are visible in the log area.

```
PL/I     LOCATION: SCLS102 :> 49.1
Command ===>                                                  Scroll ===> PAGE
MONITOR --+----1----+----2----+----3----+----4----+----5----+----6 LINE: 1 OF 3
****************************** TOP OF MONITOR *******************************
0001  1 SQLCODE                0
0002  2 QSTAFF.ID           20
0003  3 QSTAFF.NAME   'PERNAL'
***************************** BOTTOM OF MONITOR *****************************



SOURCE: SCLS102 --1----+----2----+----3----+----4----+----5--- LINE: 176 OF 195
      48   |END;                                                              .
           |                                                                  .
      49   |  CSQLCODE = SQLCODE;                                             .
      50   |  IF CSQLCODE = 0 THEN DO;                                        .
      51   |     CNAME   = QSTAFF.NAME;                                       .
      52   |     CDEPT   = QSTAFF.DEPT;                                       .
      53   |     CJOB    = QSTAFF.JOB;                                        .
      54   |     CYEARS  = QSTAFF.YEARS;                                      .
LOG 0----+----1----+----2----+----3----+----4----+----5----+---- LINE: 10 OF 18
0010    LIST SQLCODE ;
0011  GO ;
0012  STEP ;
0013  AT 49 ;
0014  GO ;
0015  MONITOR LOCAL
0016    LIST QSTAFF.ID ;
0017  MONITOR LOCAL
0018    LIST QSTAFF.NAME ;
PF  1:?          2:STEP       3:QUIT       4:LIST       5:FIND       6:AT/CLEAR
PF  7:UP         8:DOWN       9:GO        10:ZOOM      11:ZOOM LOG  12:RETRIEVE
```

Figure 86. MFI Debug Tool: Snapshot during Execution

MFI Debug Tool under CICS

As described above you must link a user options module, CEEUOPT, to the load module to activate the Debug Tool under CICS. Invoke the transaction in a CICS terminal window, and the MFI Debug Tool appears with the same window layout as described above.

Note: If you do not provide a preferences file with the location of the source listings, you may issue a SET SOURCE or PANEL SOURCE command to enter the fully qualified name of the listing file for each compile unit.

PWS Debug Tool

To bring up the PWS Debug Tool, first start the CODE/370 Debug Tool from the CODE/370 folder (see Figure 77 on page 92).

This brings up the CODE - Debug Tool Command Log window and a message box asking you to start the program on the host (see Figure 87).

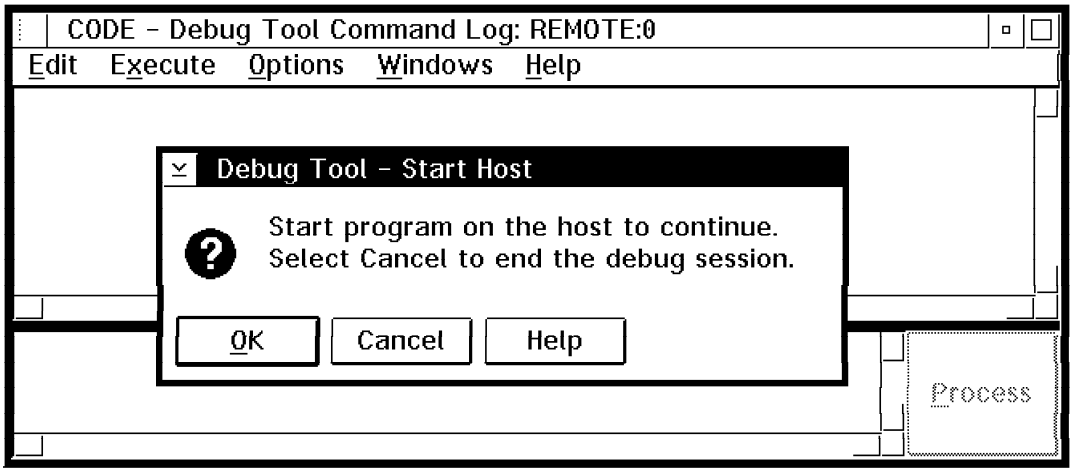


Figure 87. PWS Debug Tool Start

Click on OK, then start the program on the host with the TEST execution time option set to indicate the PWS debugger.

PWS Debug Tool under CICS

By linking a user options module, CEEUOPT, with proper execution options to the CICS transaction program you can invoke the PWS Debug Tool for a CICS program.

For example, we linked the PLISTUB program with a CEEUOPT specifying the PWS Debug Tool and started the TCLL transaction in a CICS terminal window. The PWS Debug Tool starts up and displays five windows on the OS/2 desktop as shown in Figure 88 on page 107.

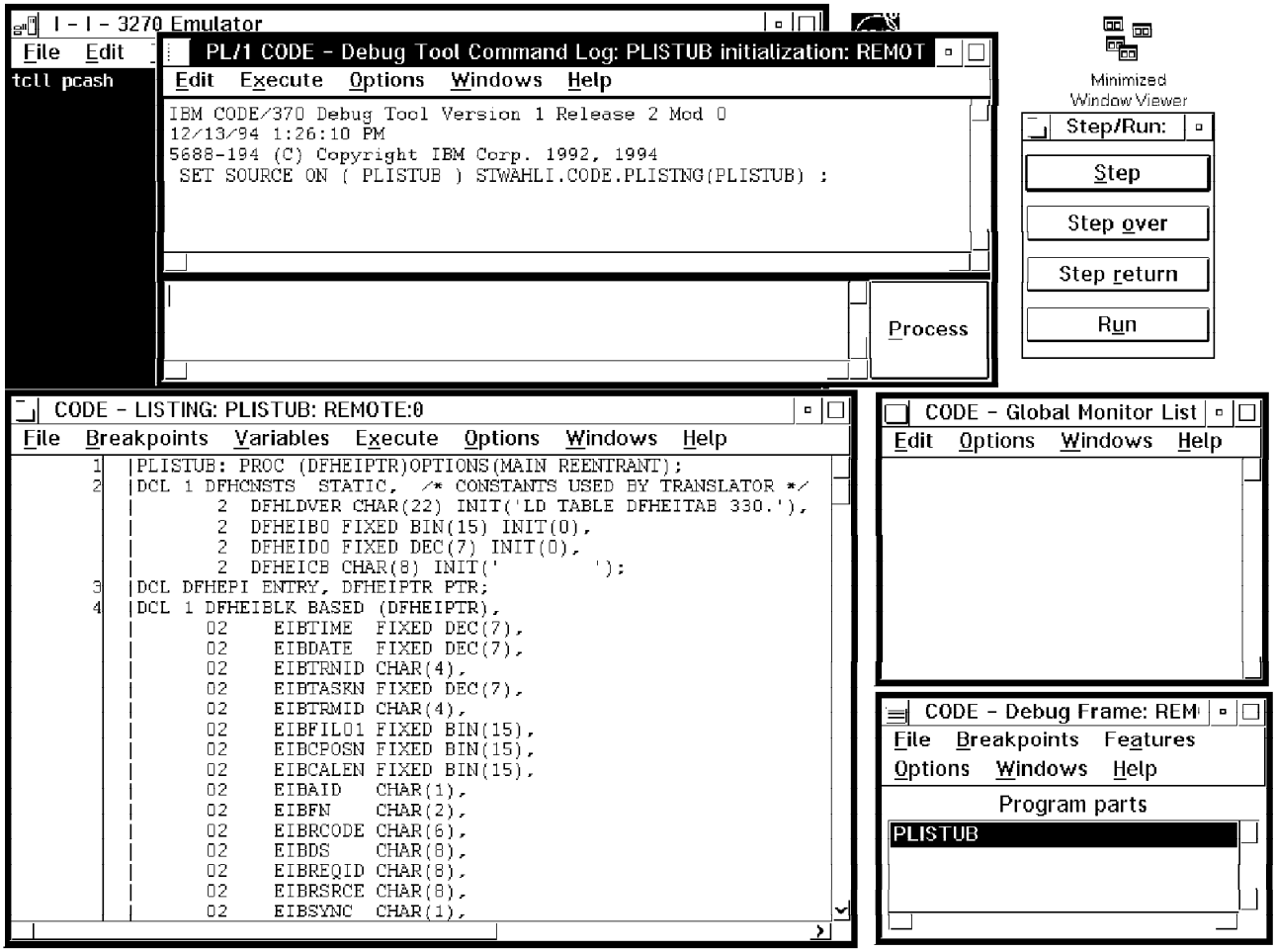


Figure 88. PWS Debug Tool Windows: Program Start

The five windows are:

- The Command Log window, which shows all commands that are executed. The bottom of the window is an input area for user commands.
- The Listing window, which shows the source listing. Here you can set breakpoints and select variables and display their values or add them to the monitor window.
- The Step/Run window, which allows you to control the execution of the program. Just click on the push buttons to step through the program.
- The Global Monitor List window, which shows the values of selected variables during execution. Values can be changed and passed back to the program.
- The Debug Frame window, which shows the current compile units. It allows you to open the listing in different formats.

In the background you can also see the 3270 Emulator window that was used as a CICS terminal window to start the TCLL transaction.

Debugging the Program

Typical activities when debugging your program are:

- Step through the program statement by statement using the Step push button.
- Set breakpoints in the program by double-clicking on a line number in the listing. Then use the Run push button to let the program run until the next breakpoint.
- Add variables to the global monitor by using a command in the Command Log window or by highlighting a variable in the Listing window with the mouse and selecting the Variables pull-down as shown in Figure 89. While you step or run through the program the Debug Tool adjusts your Listing window and updates values in the Global Monitor window automatically.

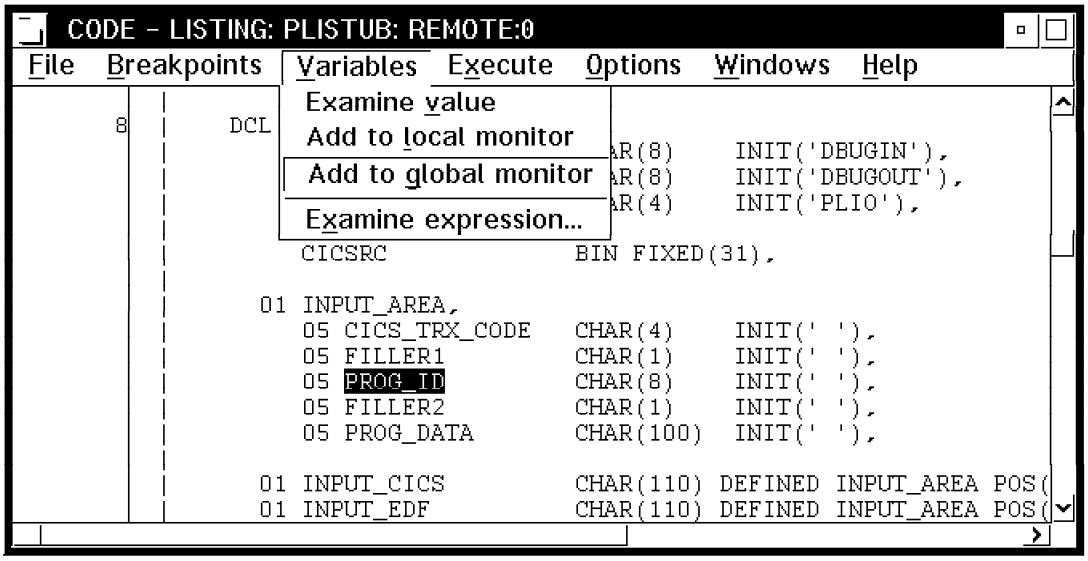


Figure 89. PWS Debug Tool: Add Variables to Global Monitor

After you step through a few statements of the program, set two breakpoints in the listing, and add two variables to the monitor, the Debug Tool shows the current state of the program as shown in Figure 90 on page 109.

These are just a few of the facilities of the PWS Debug Tool.

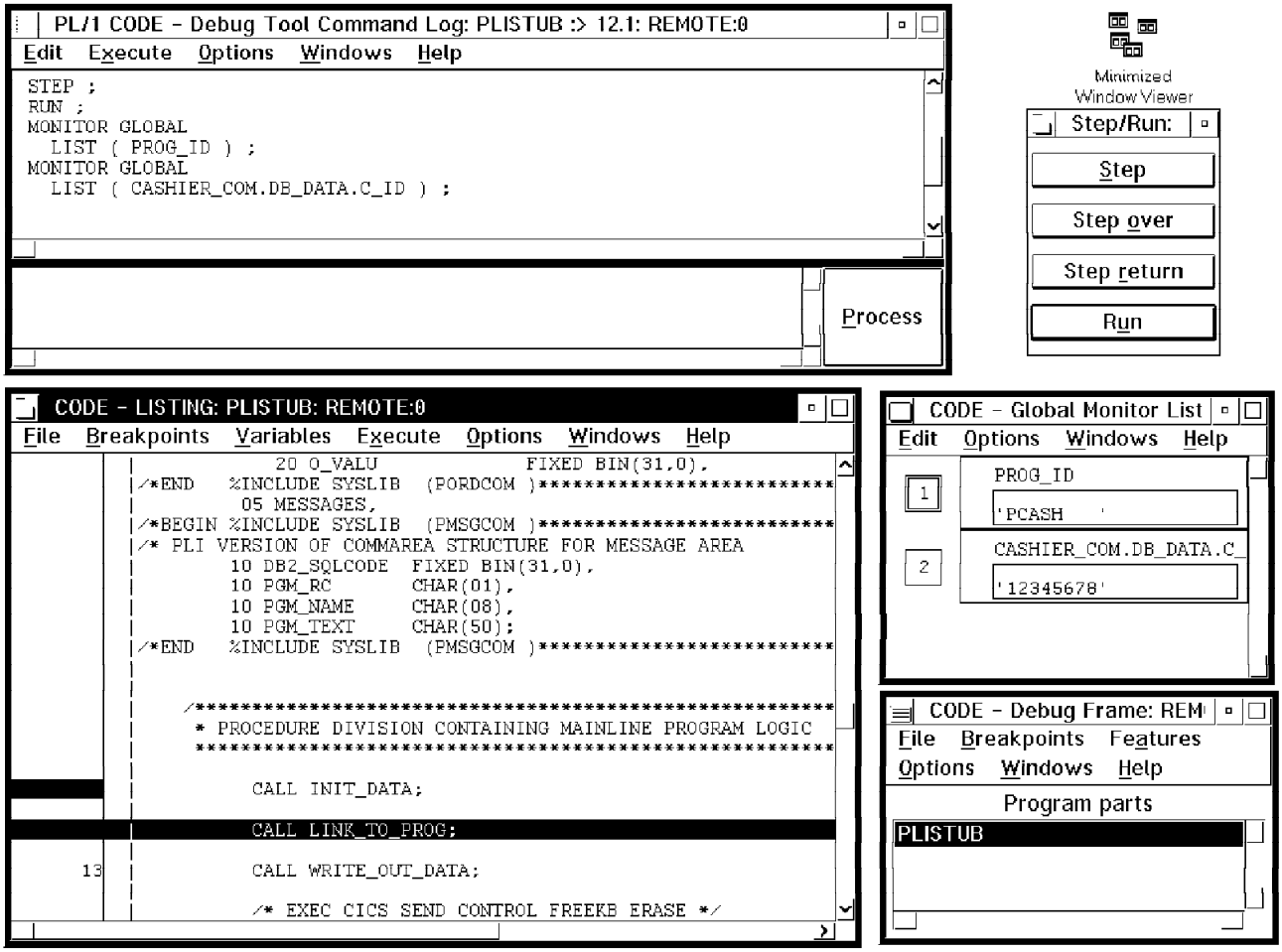


Figure 90. PWS Debug Tool Windows: Execution Snapshot

Window List

Using CODE/370 on the workstation involves working with many different windows. CODE/370 provides the Window List window, accessible from most other windows through the Windows pull-down.

The Window List window shown in Figure 91 on page 110 lists all of the CODE/370 windows and can be used to find and select any of the currently hidden windows, both debug and editor windows.

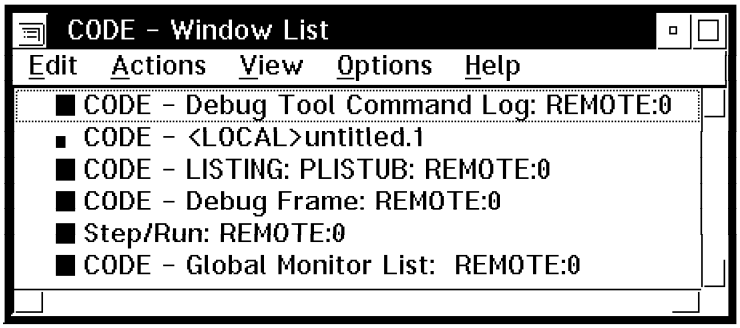


Figure 91. CODE/370 Window List

The Window List window is the only CODE/370 window showing up in the OS/2 task list, which is displayed using Control-Escape keys. From the Window List you can then find all other CODE/370 windows.

CODE/370 and WorkFrame/2

WorkFrame/2 Version 2.5 is shipped as part of CODE/370. In this section we look at how to use WorkFrame/2 with CODE/370.

After installation of the WorkFrame/2 component of CODE/370 you find an IBM WorkFrame/2 V2.5 folder (see Figure 60 on page 79), a Projects folder and an Actions Profile folder in the CODE/370 folder (see Figure 77 on page 92), and several new templates, such as an MVS Project, in the OS/2 Templates folder.

Before starting with WorkFrame/2, review the online manual “CODE, WF Integration Guide” in the CODE Information folder.

Creating a Project

To create a new project for working with an MVS PL/I application either drag the MVS Project from the OS/2 Templates folder or open the Projects folder of CODE/370 and copy the MVS Project.

Name the new project, for example, MVS PL/I Redbook Project, as shown in Figure 92.

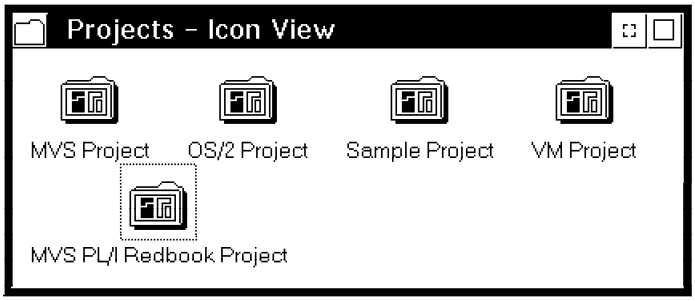


Figure 92. CODE/370 WorkFrame/2 Projects

Open the settings of the project to enter the file mask of source files. Using the naming convention described in “Host Naming Convention” on page 93, we entered <codeg>userid.CODE.* as a mask for data set names (see Figure 93 on page 111). <codeg> is the name of the CODE/370 server.

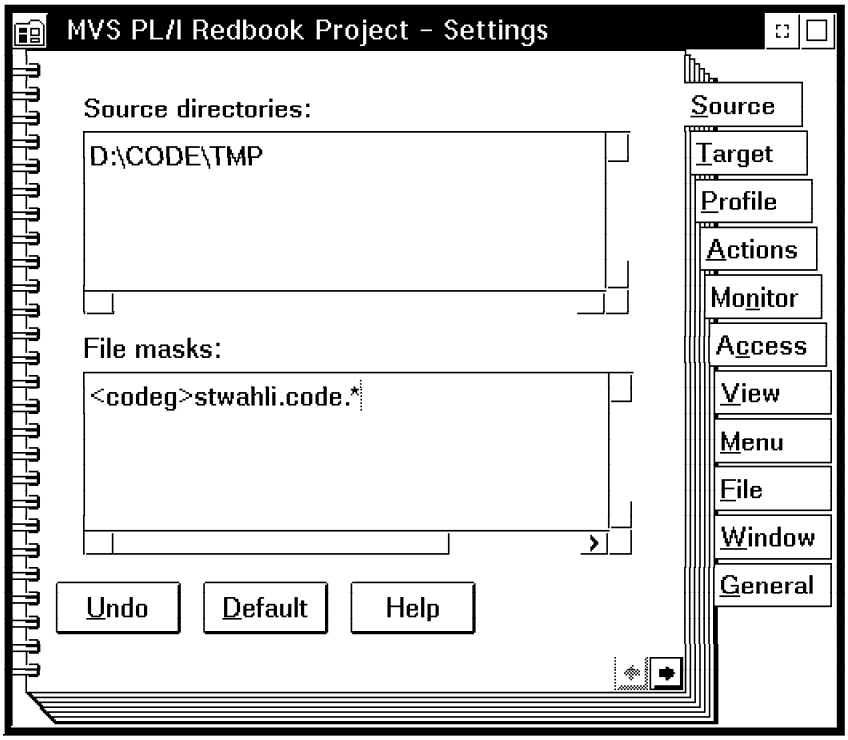


Figure 93. MVS PL/I Redbook Project Settings

Be sure to have the server started on MVS before proceeding (EVFSTART command in TSO).

Editing Source Programs

Open the project (double-click) to get a list of all data sets with the requested file mask as shown in Figure 94 on page 112.

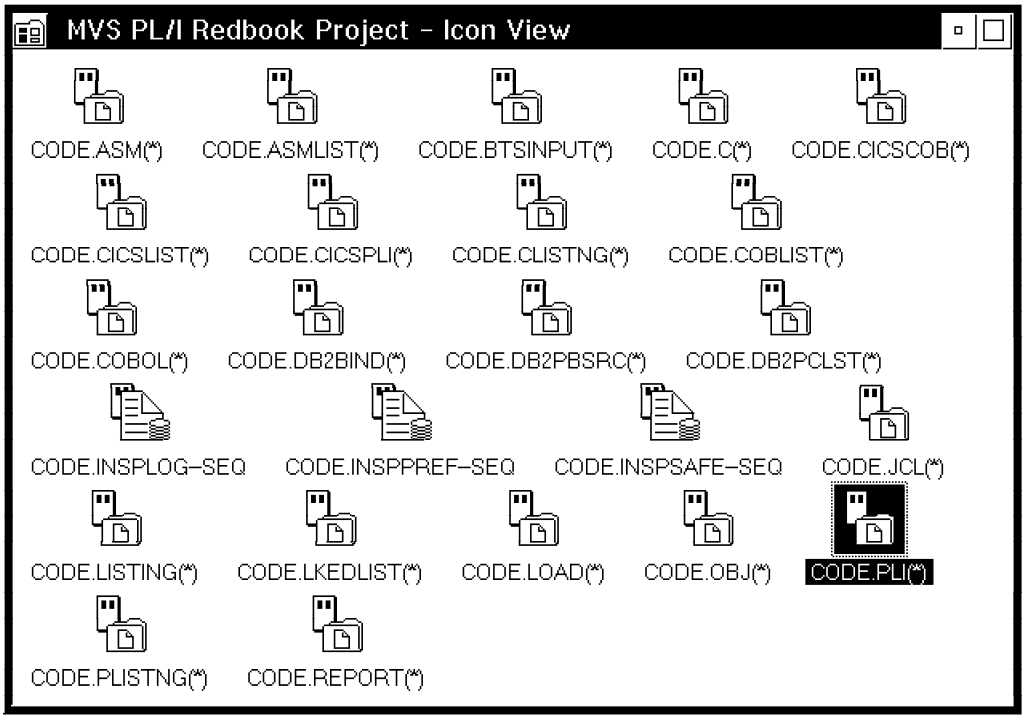


Figure 94. MVS PL/I Redbook Project Folder

Note: Instead of the Icon view you can also open the tree or details view.

Open the PL/I source library, CODE.PLI(*), to get a folder of all members in the data set. Figure 95 shows the folder with the pop-up menu for one of the members.

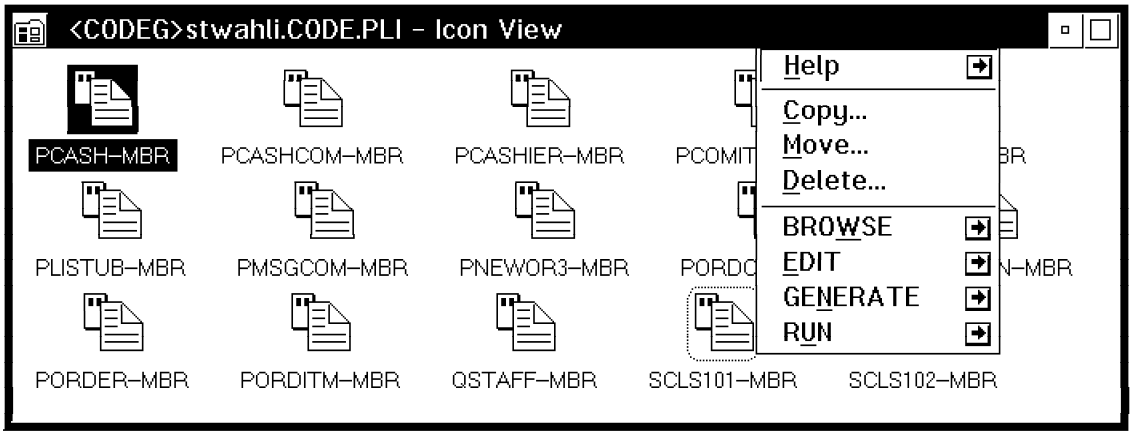


Figure 95. MVS PL/I Source Code Folder

To edit the PCASH member use the pop-up menu and select EDIT. An LPEX window is opened, the source code is downloaded, and you will be in an LPEX edit session as shown in Figure 80 on page 95.

Compile and Link

Use the pop-up menu for a member and select the arrow to the right of GENERATE and then Program... in the submenu, and the Program Generator window opens as shown in Figure 82 on page 99.

Submit the compile and link operations with proper parameters and options.

Tailoring an Action

To invoke the Program Generator with the correct action we added a PL/I compile action to the Actions Profile. This is the same process as in “Adding Own Actions” on page 89.

We opened the project settings, went to the Profile page, and opened the MVS Actions Profile (see Figure 96). We browsed the current GENERATE actions to look at the parameters. Then we added the new action PL/I Compile under GENERATE with the additional parameter /A PLI_COMPILE_PP, which is the command procedure we wanted to invoke at the host.

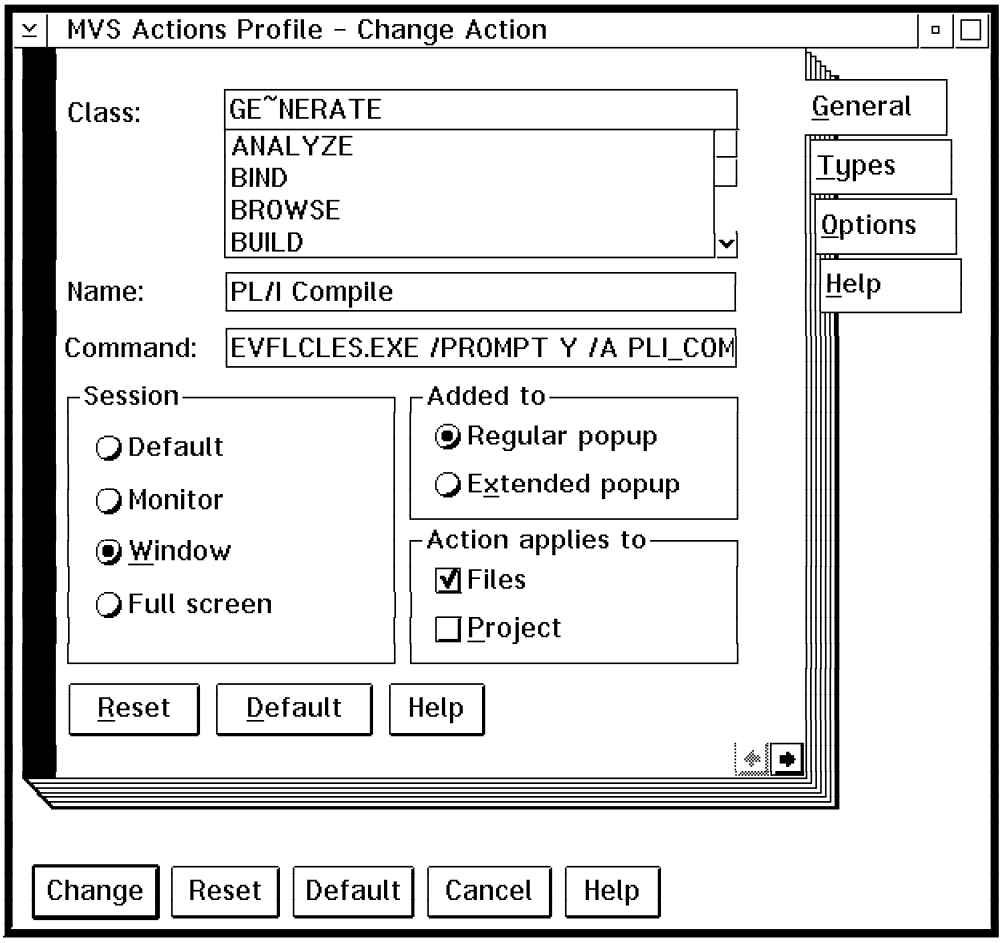


Figure 96. Tailoring the GENERATE Action for PL/I

Using the tailored GENERATE action will set the correct compile procedure when the Program Generator window opens.

CODE/370 and WorkFrame/2 Integration

WorkFrame/2 provides an integrated edit, compile, and debug environment for CODE/370. In the previous few pages we only describe the very basic possibilities. An advanced user will find much more flexibility in the combination of the two products.

Appendix A. Sample Code

This appendix contains sample code developed during the project.

REXX Program to Run the DBM Command Processor

The RUNSQL program reads SQL statements from an MVS DB2 SPUFI file and invokes the DBM command processor of DB2/2 for each statement.

Statements are condensed by eliminating blanks to fit within 256 bytes (if possible), which is the limit for an OS/2 command.

Program RUNSQL

```
/* Rexx - read a SPUFI file and submit SQL statements using DBM */

   arg file .                              /* input file */
   stat = STREAM(file,'C','open read')     /* open it */
   if left(stat,5) <> 'READY' then
      call error 'Cannot open SQL file: ' file
   stmt = ''                               /* build statement */
   complete = 0
   do while lines(file)>0
        line = linein(file)
        line = strip(line,'T')
        if left(line,2)='--' then iterate  /* ignore comments */
        lg = length(line)
        if lg>0 then                       /* check for ; */
           if substr(line,lg,1) = ';' then do
              complete = 1
              line = left(line,lg-1)
           end
        if stmt='' then stmt = left(line,72)' '
                   else stmt = stmt''left(line,72)' '
        if complete then do                /* submit complete stmt */
           say
           say 'Executing:'                /* display statement */
           do i=1 by 73 to length(stmt)
              say '    ' substr(stmt,i,73)
           end
           stmt = space(stmt)              /* remove extra blanks */
           "@CALL DBM -R(runsql.lst)" stmt  /* call DBM procesoor */
           stmt = ''
           complete = 0
        end
   end /* do */
```

Figure 97 (Part 1 of 2). Listing of Program RUNSQL.CMD

```
   stat = STREAM(file,'C','close')          /* close file */
   return
error:                                      /* error messages */
   parse arg msg
   say 'RUNSQL ERROR:' msg
   exit 8
```

Figure 97 (Part 2 of 2). Listing of Program RUNSQL.CMD

PL/I for OS/2 Compile and Link Procedures

Some compile and link procedures are shipped with PL/I for OS/2 and CICS 2.0.

The combination of both DB2 and CICS access from one PL/I program is not covered by either product; therefore we developed additional compile and link procedures.

Program PLIPPENV - Set PL/I Environment Variables

This procedure sets environment variables for the DB2 SQL preprocessor and optionally for the CICS preprocessor.

```
/*********************************************************************/
/* Rexx cmd file to set preprocessor options                         */
/*********************************************************************/
   os2 = 'OS2ENVIRONMENT'

   arg database sqlopt '(' cicsopt .
   if database = '' then database = 'SAMPLE'

   db2   = 'DBNAME('database') ISOLATION(CS) BIND' sqlopt
   cics  = cicsopt

   db2  = value('IBM.PPSQL',db2,os2)
   cics = value('IBM.PPCICS',cics,os2)

   incl  = value('INCLUDE',,os2)
   if pos('PLIHDR',incl)=0 then
      incl = value('INCLUDE','D:\CICS200\PLIHDR;'incl,os2)

   "@SET INCLUDE"
   "@SET IBM.PPSQL"
   "@SET IBM.PPCICS"
```

Figure 98. Listing of Program PLIPPENV.CMD

Program PLILINKP - Link a PL/I Program with SQL/CICS

This procedure links a PL/I program that contains both DB2 and CICS statements into an executable (.EXE). The procedure is a copy of PLILINK; additions and changes are marked with an ITSO comment.

```
/*********************************************************************/
/* Rexx cmd file to link a PL/I program with SQL and CICS          */
/*********************************************************************/

 os2 = 'OS2ENVIRONMENT'                                  /* ITSO */
 cics = value('CICSSET',,os2)                            /* ITSO */
 if cics='' then "@call CICSENV"                         /* ITSO */

parse upper arg string

/* setting the defaults */
deffile = 'nul.def'
Vmode   = 'VIO'
os2lib  = ''
db2lib  = '+sql_dyn'                                     /* ITSO */
cicslib = '+faapstrt+faaplid+faaotspl+faacic32'          /* ITSO */

/* processing parms */
if pos('.DEF', string) > 0 then
  do
    deffile = word(string, words(string))
    posdef  = pos(deffile, string)
    string  = left(string, posdef-1)
  end

ppm = pos('/PM:', string)
if ppm > 0 then
  do
    Vmode = word(substr(string, ppm+4), 1)
    if Vmode = 'PM' then
      os2lib = '+OS2386'
    string = left(string, ppm-1)
  end

/* linking */
'link386' string '/pm:'Vmode '/noe /co' ,                /* ITSO */
          '/stack:7000000,,load.map,ceelink +ibmlink' ,   /* ITSO */
          db2lib os2lib cicslib','deffile                 /* ITSO */
```

Figure 99. Listing of Program PLILINKP.CMD

Program CICSPSQC - Compile a PL/I SQL CICS Program

This procedure compiles a PL/I program with both SQL and CICS statements. It is a copy of CICSPCMD; changes are marked with an ITSO comment.

```
/* CICSpsqc.CMD */
/*********************************************************************/
/*                                                                   */
/* MODULE NAME        CICSpsqc.CMD                                   */
/*                                                                   */
/* DESCRIPTIVE NAME   Compile a PL/I program with SQL                */
/*                                                                   */
/* Statement:         Licensed Materials - Property of IBM           */
/*                                                                   */
/*                    53G3861,53G3862                                */
/*                    (c) Copyright IBM Corp. 1988, 1993.            */
/*                                                                   */
/*                    See Copyright Instructions.                    */
/*                                                                   */
/*                    All rights reserved.                           */
/*                                                                   */
/*                    U.S. Government Users Restricted Rights - use, */
/*                    duplication or disclosure restricted by GSA    */
```

Figure 100 (Part 1 of 2). Listing of Program CICSPSQC.CMD

```
/*                   ADP Schedule Contract with IBM Corp.            */
/*                                                                   */
/* Status:           Version 2 Release 0                             */
/*                                                                   */
/*  NOTES :-                                                         */
/*    DEPENDENCIES = None                                            */
/*                   None                                            */
/*    RESTRICTIONS = none                                            */
/*    MODULE TYPE  = Command file                                    */
/*    PROCESSOR    = PS/2 and PC                                     */
/*                                                                   */
/*********************************************************************/
/* Parse program names and user options */
parse arg pliprogs '('  usercompopts

/* Display help? */
if pliprogs = ''  then
   do
   call help
   return 0
   end

/* Set up CICS environment */
If GetValue('CICSSET') = '' Then '@CALL CICSENV'
If GetValue('IBM.PPSQL') = '' Then '@CALL PLIPPENV srvdb'      /* ITSO */

/* Set compiler options */
compopts = '(system(cics) pp(sql cics("deck print") macro) '||,/* ITSO */
           'tiled source not("¬ê") '||usercompopts

/* Compile programs */
saverc = 0
do j = 1 to words(pliprogs)
    "pli" word(pliprogs,j) compopts
    if rc > saverc then saverc = rc
    say
    say "Compile" word(pliprogs,j) "rc = "rc
end

return saverc

/*-------------------------------------------------------------------*/
GetValue: Procedure
  Arg EnvVal .
  res  = value(EnvVal,,'OS2ENVIRONMENT')
Return res
/*-------------------------------------------------------------------*/

help: procedure
   say '                                                                '
   say 'This command will translate and compile a CICS PL/I program.         '
   say 'The input into this process is:-
   say '                                                                '
   say '  File             Contents                  Source             '
   say '-------------    ----------------------    -----------------     '
   say 'progname.PLI     CICS Source module        user defined          '
   say
   say 'The output is:-
   say 'progname.LST     PL/I Compiler Listing                           '
   say 'progname.OBJ     Object module                                   '
   say '                                                                '
   say 'To run this command enter:
   say
   say '  CICSpsqc Pliprogs ( UserCompilerOptions'
   say
   say '  Pliprogs            - List of PL/I source programs to be translated and'
   say '                        compiled.'
   say '  UserCompilerOptions - Additional user PL/I compiler options.'
   return
```

Figure 100 (Part 2 of 2). Listing of Program CICSPSQC.CMD

REXX Program to Generate Own Code Blocks

This program reads the My Code Blocks file (PMGMYCB.PLF) and creates an application code block library.

```
/* REXX - Create Code Block Files for Visual PL/I  - ITSO San Jose  */
/*------------------------------------------------------------------*/
/* Utility to conver My Code Blocks to Application Code Blocks.     */
/*                                                                  */
/* Use it to manage your code block development environment.        */
/* It takes code blocks from My Code Block and creates 4 files as   */
/* per instructions in Appendix B of the PL/I for OS/2 Toolkit Ref. */
/* Place this file in your PL/I toolkit directory.                  */
/*                                                                  */
/* Syntax: MYCB2CB filename                                         */
/*         where filename is target name (no extension)             */
/*                                                                  */
/* You will be prompted for a description.                          */
/* If target files already exist, the program will exit.            */
/*------------------------------------------------------------------*/
signal on syntax
signal on notready

parse arg filepfx

if filepfx = "" | filepfx = '?' | pos('.',filepfx)>0 then do
   say
   say "Usage:  mycb2cb outputfile"
   say "        where outputfile is the output file prefix (no .)"
   say
   say "This exec creates outputfile.pl  - program library file"
   say "                  outputfile.plf - function file"
   say "                  outputfile.plm - module file"
   say "                  outputfile.plo - object file"
   say
   say "Converts My Code Blocks (PMGMYCB.PLF) as per"
   say "Appendix B of the PL/I for OS/2 Toolkit Reference"
   say
   say "The 4 target files can then be taken to"
   say "other Visual PL/I user's machines and read in as code blocks"
   exit
end
inputfile  = 'PMGMYCB.PLF'

                    /* get the filename and translate to lowercase */
                    /*--------------------------------------------*/
filename = strip(word(filepfx,1))
lower_alpha = 'abcdefghijklmnopqrstuvwxyz'
upper_alpha = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
filename = translate(filename,lower_alpha,upper_alpha)

                                          /* initialize variables */
                                          /*----------------------*/
xPL  = ''
xPLF = ''
xPLM = ''
xPLO = ''
dot_object = '&&' || filename || '.obj'

                         /* check if source and target files exist */
                         /*----------------------------------------*/
xIN  = stream(inputfile,       'c','query exists')
xPL  = stream(filename||'.PL ','c','query exists')
xPLF = stream(filename||'.PLF','c','query exists')
xPLM = stream(filename||'.PLM','c','query exists')
xPLO = stream(filename||'.PLO','c','query exists')
if xIN   = '' then say 'Input' inputfile 'does not exist'
```

Figure 101 (Part 1 of 2). Listing of Code Block Generator MYCB2CB.CMD

```
if xPL  <> '' then say 'You already have' xPL
if xPLF <> '' then say 'You already have' xPLF
if xPLM <> '' then say 'You already have' xPLM
if xPLO <> '' then say 'You already have' xPLO
if xIN='' | (xPL || xPLF || xPLM || xPLO)<> '' then exit 8

                                    /* get a one line description */
                                    /*----------------------------*/
functionality = ''
do while (functionality = '')
   say 'please enter a one line description for code blocks:'
   parse pull functionality
   functionality = strip(functionality)
end
                               /* write the Program Library file .PL */
                               /*------------------------------------*/
outputfile = filename||'.pl'
rc = stream(outputfile,'c','OPEN')
rc = lineout(outputfile, "Module   " filename||'.plm')
rc = lineout(outputfile, "Objects  " filename||'.plo')
rc = lineout(outputfile, "Functions" filename||'.plf')
rc = stream(outputfile,'c','CLOSE')
                                    /* write the Module file .PLM */
                                    /*----------------------------*/
outputfile = filename||'.plm'
rc = stream(outputfile,'c','OPEN')
rc = lineout(outputfile, functionality)
rc = lineout(outputfile, " " || dot_object)
rc = stream(outputfile,'c','CLOSE')
                                    /* write the Module file .PLO */
                                    /*----------------------------*/
outputfile = filename||'.plo'
rc = stream(outputfile,'c','OPEN')
rc = lineout(outputfile, dot_object)
rc = stream(outputfile,'c','CLOSE')
                                /* write the Function file .PLF      */
                                /* copy code blocks from PMGMYCB.PLF */
                                /* modifying the &&pmgmycb.obj names */
                                /*-----------------------------------*/
outputfile = filename || '.PLF'
rc = stream(inputfile, 'c','OPEN')
rc = stream(outputfile,'c','OPEN')
line_num = 0
do while lines(inputfile)
   line = linein(inputfile)
   line_num = line_num + 1
   if substr(line,1,20) == '&&pmgmycb.obj BEGIN ' then
      outputline = dot_object subword(line,2)
   else
      outputline = line
   rc = lineout(outputfile,outputline)
end
rc = stream(inputfile, 'c','CLOSE')
rc = stream(outputfile,'c','CLOSE')

exit
```

Figure 101 (Part 2 of 2). Listing of Code Block Generator MYCB2CB.CMD

ISPF/DB2 Application

The PL/I programs of the ISPF/DB2 application are presented below.

Program SCLS101 Original

This is the original PL/I ISPF main program on MVS.

```
/*********************************************************************/
/* SCLS101 Main - ISPF USER INTERFACE                                */
/*********************************************************************/

SCLS101:  PROC OPTIONS(MAIN) ;

/*********************************************************************/
/*     External Entries  and  Builtin Functions                      */
/*********************************************************************/
DCL FLMLNK          ENTRY EXTERNAL OPTIONS(ASM,INTER,RETCODE) ;
DCL ISPLINK         ENTRY EXTERNAL OPTIONS(ASM,INTER,RETCODE) ;
DCL SCLS102         ENTRY EXTERNAL ;
DCL PLIRETV         BUILTIN ;

/*********************************************************************/
/*     Other Variable Declarations                                   */
/*********************************************************************/
DCL
     TSQLCODE        BIN FIXED(31)      INIT(0),      /* SQL */
     TID             BIN FIXED(15)      INIT(0),      /* TABLE */
     TNAME           CHAR(9) VAR        INIT(' '),
     TDEPT           BIN FIXED(15)      INIT(0),
     TJOB            CHAR(5)            INIT(' '),
     TYEARS          BIN FIXED(15)      INIT(0),
     TSALARY         FIXED DECIMAL(7,2) INIT(0),
     TCOMM           FIXED DECIMAL(7,2) INIT(0),
     CXYZ            CHAR(5)            INIT(' '),    /* ISPF */
     CID             CHAR(5)            INIT(' '),
     CNAME           CHAR(9)            INIT(' '),
     CDEPT           CHAR(5)            INIT(' '),
     CJOB            CHAR(5)            INIT(' '),
     CYEARS          CHAR(5)            INIT(' '),
     CSALARY         CHAR(8)            INIT(' '),
     CCOMM           CHAR(8)            INIT(' '),
     MSGLINE         CHAR(79)           INIT(' '),
     LXYZ            BIN FIXED(31)      INIT(5),      /* LENGTHS */
     LID             BIN FIXED(31)      INIT(5),
     LNAME           BIN FIXED(31)      INIT(9),
     LDEPT           BIN FIXED(31)      INIT(5),
     LJOB            BIN FIXED(31)      INIT(5),
     LYEARS          BIN FIXED(31)      INIT(5),
     LSALARY         BIN FIXED(31)      INIT(8),
     LCOMM           BIN FIXED(31)      INIT(8),
     LMSGLINE        BIN FIXED(31)      INIT(79),
     PIC5            PICTURE '99999',                 /* CONVERSION */
     PIC5Z           PICTURE 'ZZZZ9',
     PIC8            PICTURE '----9.V99',
     PIC6S           PICTURE 'SZZZZ9',
     RETCODE         BIN FIXED(15);                   /* RETURNCODE */

/*********************************************************************/
/*     ISPF Definitions                                              */
/*********************************************************************/

CALL ISPLINK ('VDEFINE ','CXYZ ',CXYZ , 'CHAR ',LXYZ);
CALL ISPLINK ('VDEFINE ','CID ',CID , 'CHAR ',LID) ;
CALL ISPLINK ('VDEFINE ','FNAME ',CNAME , 'CHAR ' ,LNAME);
CALL ISPLINK ('VDEFINE ','CDEPT ',CDEPT , 'CHAR ',LDEPT) ;
```

Figure 102 (Part 1 of 2). Listing of PL/I Program SCLS101

```
CALL ISPLINK (’VDEFINE ’,’JOB ’,CJOB , ’CHAR ’,LJOB) ;
CALL ISPLINK (’VDEFINE ’,’CYEARS ’,CYEARS ,’CHAR ’,LYEARS) ;
CALL ISPLINK (’VDEFINE ’,’CSALARY ’,CSALARY,’CHAR ’,LSALARY) ;
CALL ISPLINK (’VDEFINE ’,’CCOMM ’,CCOMM ,’CHAR ’,LCOMM) ;
CALL ISPLINK (’VDEFINE ’,’MSG ’,MSGLINE , ’CHAR ’ ,LMSGLINE) ;

/*********************************************************************/
/****   MAIN PROGRAM LOGIC                                        ****/
/*********************************************************************/

CALL ISPLINK (’DISPLAY ’,’SCLPS001’) ;              /* DISPLAY PANEL */
RETCODE = PLIRETV() ;

DO WHILE (RETCODE = 0) ;
  PIC5 = CXYZ ;                                     /* ID FROM PANEL */
  TID  = PIC5;
  CID  = CXYZ ;

  CALL SCLS102 ( TID, TNAME, TDEPT, TJOB, TYEARS,   /* CALL SUBROUT. */
                      TSALARY, TCOMM, TSQLCODE ) ;  /* FOR DB2 DATA  */

  CALL PARMV ;                                      /* ISPF VARIABLES*/

  CALL ISPLINK (’DISPLAY ’,’SCLPS001’) ;            /* DISPLAY PANEL */
  RETCODE = PLIRETV() ;
END ;

PARMV: PROC ;
     PIC5    = TID;        CID     = PIC5;  CXYZ = CID;
  IF TSQLCODE = 0 THEN DO;
     CNAME   = TNAME;
     CJOB    = TJOB;
     PIC5Z   = TDEPT;      CDEPT   = PIC5Z;
     PIC5Z   = TYEARS;     CYEARS  = PIC5Z;
     PIC8    = TSALARY;    CSALARY = PIC8;
     PIC8    = TCOMM;      CCOMM   = PIC8;
     MSGLINE = ’O.K., THIS IS YOUR DATA’ ;
     END;
  ELSE DO;
     CNAME   = ’--none--’;
     CJOB    = ’ ’;
     CDEPT   = ’ ’;
     CYEARS  = ’ ’;
     CSALARY = ’ ’;
     CCOMM   = ’ ’;
     PIC6S   = TSQLCODE;
     MSGLINE = ’REQUEST FAILED FOR THIS ID, SQLCODE = ’||PIC6S ;
  END;
END PARMV ;

END SCLS101 ;
```

Figure 102 (Part 2 of 2). Listing of PL/I Program SCLS101

Program SCLS102

This is the original PL/I DB2 subroutine on MVS. The additional CONNECT statement for DB2/2 is shown as a comment.

```
/*********************************************************************/
/* SCLS102 Subroutine - Retrieve Row from STAFF table                */
/*********************************************************************/
```

Figure 103 (Part 1 of 2). Listing of Program SCLS102

```
SCLS102: PROC ( CID, CNAME, CDEPT, CJOB, CYEARS,
                CSALARY, CCOMM, CSQLCODE );

dcl cid             bin fixed(15),
    cname           char(9) var,
    cdept           bin fixed(15),
    cjob            char(5),
    cyears          bin fixed(15),
    csalary         fixed decimal(7,2),
    ccomm           fixed decimal(7,2),
    csqlcode        bin fixed(31);

/*********************************************************************/
/* Include DCLGEN output                                             */
/*********************************************************************/
  EXEC SQL INCLUDE QSTAFF ;
  qstaff.id = cid;

/*********************************************************************/
/* Include SQL communication area                                    */
/*********************************************************************/
  EXEC SQL INCLUDE SQLCA ;

/*********************************************************************/
/* Connect to Sample Database - Workstation only                     */
/*********************************************************************/
/* EXEC SQL CONNECT TO SAMPLE ; */

/*********************************************************************/
/* Retrieve row from STAFF table                                     */
/*********************************************************************/
  EXEC SQL SELECT ID, NAME, DEPT, JOB, YEARS, SALARY, COMM
             INTO :QSTAFF
             FROM Q.STAFF
            WHERE ID = :QSTAFF.ID ;

  csqlcode = SQLCODE;
  if csqlcode = 0 then do;
     cname   = qstaff.name;
     cdept   = qstaff.dept;
     cjob    = qstaff.job;
     cyears  = qstaff.years;
     csalary = qstaff.salary;
     ccomm   = qstaff.comm;
     end;
  else do;
     cname   = '--none--';
     cdept   = 0;
     cjob    = '';
     cyears  = 0;
     csalary = 0;
     ccomm   = 0;
  end;

END SCLS102 ;
```

Figure 103 (Part 2 of 2). Listing of Program SCLS102

CICS/DB2 Application

The PL/I source listings of the CICS DB2 application are presented below.

Program PCASH

This program retrieves cashier information from the DB2 cashier table and passes it back in the CICS communications area.

```
PCASH: PROC (COMMAREA_PTR) OPTIONS(MAIN REENTRANT);
/*****************************************************************/
/* ITSC - SAN JOSE   CLIENT/SERVER COMPUTING USING DCE/AS        */
/*                                                               */
/* DESCRIPTION:  PLI VERSION OF CCASHIER COBOL PROGRAM TO GET    */
/*               CASHIER DETAILS FOR A SPECIFIC 'CASHIER ID'     */
/*                                                               */
/*               WHEN A CASHIER INITIALLY LOGS ON, THE CLIENT    */
/*               QUERIES THE SERVER DATABASE TO VALIDATE THAT THE*/
/*               CASHIER EXISTS.                                 */
/*                                                               */
/* INPUT:  'CASHIER ID' FROM CLIENT APPLICATION                  */
/*                                                               */
/* OUTPUT:  ONE ROW OF DATA FROM SRV.CASHIER DB2 TABLE           */
/*****************************************************************/

/*****************************************************************/
/* SQL INCLUDES AND DEFINITIONS                                  */
/*****************************************************************/

/* DB2 SQL COMMUNICATIONS AREA  */
EXEC SQL INCLUDE SQLCA;

/* DCLGEN MEMBER FOR CASHIER TABLE */
EXEC SQL INCLUDE PCASHIER;

/*****************************************************************/
/* COMMON PROGRAM VARIABLES USED BY ALL SERVER PROGRAMS          */
/*****************************************************************/

%INCLUDE PCOMITMS;

/*****************************************************************/
/*  DECLARATIONS FOR PLI BUILTIN FUNCTIONS                       */
/*****************************************************************/

DCL ADDR BUILTIN;
DCL NULL BUILTIN;

/*****************************************************************/
/* COMMAREA USED TO PASS DATA BETWEEN CLIENT AND SERVER          */
/*****************************************************************/

DCL COMMAREA_PTR PTR;

DCL 01 COMMAREA BASED(COMMAREA_PTR) UNALIGNED,
      05 DB_DATA,
        %INCLUDE PCASHCOM;
      05 MESSAGES,
        %INCLUDE PMSGCOM;

/*****************************************************************/
/* MAIN ROUTINE CONTAINING MAINLINE PROGRAM LOGIC                */
/*****************************************************************/

CALL INIT_DATA;
```

Figure 104 (Part 1 of 2). Listing of Program PCASH

```
CALL GET_DATA;

CALL PROCESS_DATA;

EXEC CICS RETURN;

/*********************************************************************/
/* SUBROUTINE TO INITIALIZE PROGRAM DATA AREAS                       */
/*********************************************************************/

INIT_DATA: PROC;

  PGM_NAME   = 'PCASHIER';
  CASHIER_ID = C_ID;

  EXEC SQL CONNECT TO SRVDB; /****************************** ITSO */

END INIT_DATA;

/*********************************************************************/
/* SUBROUTINE TO RETRIEVE DATA INTO DCLGEN AREA                      */
/*********************************************************************/

GET_DATA: PROC;

  EXEC SQL SELECT CASHIER_NAME, BRANCH_CODE
           INTO :CASHIER_NAME, :BRANCH_CODE
           FROM SRV.CASHIER
           WHERE CASHIER_ID   = :CASHIER_ID;

END GET_DATA;

/*********************************************************************/
/* SUBROUTINE TO MOVE DCLGEN AREA INTO COMMAREA                      */
/*********************************************************************/

PROCESS_DATA: PROC;

  DB2_SQLCODE = SQLCODE;

  IF SQLCODE = SQL_GOOD THEN
    DO;
      C_NAME   = CASHIER_NAME;
      B_CODE   = BRANCH_CODE;
      PGM_RC   = PGM_SUCCESS;
      PGM_TEXT = 'SUCCESSFUL PROGRAM COMPLETION';
    END;
  ELSE
    IF SQLCODE = SQL_NO_DATA THEN
      DO;
        PGM_RC   = PGM_NO_DATA;
        PGM_TEXT = 'NO DATA FOR THIS CASHIER ID';
      END;
    ELSE
      DO;
        PGM_RC   = PGM_FAILURE;
        PGM_TEXT = 'BAD SQLCODE ON SELECT CASHIER DATA';
      END;

END PROCESS_DATA;

END PCASH;
```

Figure 104 (Part 2 of 2). Listing of Program PCASH

Program PNEWOR3

This program updates the two tables of order information based on a data structure in the CICS communications area.

```
PNEWOR3: PROC (COMMAREA_PTR) OPTIONS(MAIN REENTRANT);
/*****************************************************************/
/* ITSC - SAN JOSE   CLIENT/SERVER COMPUTING USING DCE/AS        */
/*                                                               */
/* DESCRIPTION:  PLI VERSION OF CNEWORD3 COBOL PROGRAM TO SAVE   */
/*               NEW AND COMPLETE ORDERS (DEMONSTRATES MAPPING A */
/*               PLI CICS COMMAREA TO A COBOL-TYPE XDL FILE)     */
/*                                                               */
/*               AFTER AN ORDER IS INITIALIZED (ORD STATUS='INI'), */
/*               IT GETS POPULATED WITH ORDER ITEM, DENOMINATION, */
/*               ORDER VALUE, AND CUSTOMER INFORMATION.          */
/*               THE ORDER IS THEN SAVED AS A NEW ORDER          */
/*               (ORD STATUS='NEW').  WHEN A CASHIER HAS FINISHED */
/*               WORKING ON A NEW ORDER, HE WILL SAVE IT AS A    */
/*               COMPLETE ORDER (ORD STATUS='COM').              */
/*               THIS PROGRAM SAVES 'NEW' AND 'COMPLETE' ORDERS. */
/*                                                               */
/* INPUT:  NEW, OR COMPLETE, ORDER DETAILS FROM CLIENT           */
/*                                                               */
/* OUTPUT:  SAVE ROWS OF DATA TO THE FOLLOWING DB2 TABLES:       */
/*          SRV.ORDER, SRV.ORDER_ITEM, SRV.ORDER_DENOM           */
/*****************************************************************/

/*****************************************************************/
/* SQL INCLUDES AND DEFINITIONS                                  */
/*****************************************************************/

/* DB2 SQL COMMUNICATIONS AREA */
EXEC SQL INCLUDE SQLCA;

/* DCLGEN MEMBER FOR ORDER TABLE */
EXEC SQL INCLUDE PORDER;

/* DCLGEN MEMBER FOR ORDER_ITEM TABLE */
EXEC SQL INCLUDE PORDITM;

/* DCLGEN MEMBER FOR ORDER_DENOM TABLE */
EXEC SQL INCLUDE PORDDEN;

/*****************************************************************/
/* COMMON PROGRAM VARIABLES USED BY ALL SERVER PROGRAMS          */
/*****************************************************************/

%INCLUDE PCOMITMS;

/*****************************************************************/
/*  DECLARATIONS FOR PLI BUILTIN FUNCTIONS                       */
/*****************************************************************/

DCL ADDR BUILTIN;
DCL NULL BUILTIN;

/*****************************************************************/
/* PROGRAM-SPECIFIC DECLARATIONS                                 */
/*****************************************************************/

DCL MAX_NUM_ITEM    FIXED BIN(15) INIT(10);
DCL MAX_NUM_DEN     FIXED BIN(15) INIT(9);
DCL TRUE_T          CHAR(1)       INIT('T');
DCL FALSE_F         CHAR(1)       INIT('F');

DCL CNT_ITEM        FIXED BIN(15);
DCL NO_MORE_ITEM    CHAR(1);
DCL QUIT_ITEM_LOOP  CHAR(1);
```

Figure 105 (Part 1 of 5). Listing of Program PNEWOR3

```
DCL CNT_DEN         FIXED BIN(15);
DCL NO_MORE_DEN     CHAR(1);
DCL QUIT_DEN_LOOP   CHAR(1);

/*********************************************************************/
/* COMMAREA USED TO PASS DATA BETWEEN CLIENT AND SERVER              */
/*********************************************************************/

DCL COMMAREA_PTR PTR;

DCL 01 COMMAREA BASED(COMMAREA_PTR) UNALIGNED,
      05 DB_DATA,
        %INCLUDE PORDCOM;
      05 MESSAGES,
        %INCLUDE PMSGCOM;

/*********************************************************************/
/* PROCEDURE DIVISION CONTAINING MAINLINE PROGRAM LOGIC              */
/*                                                                   */
/* FIRST DELETE ALL THE EXISTING ORDER ITEMS AND DENOMINATIONS.      */
/* THEN, UPDATE THE ORDER TABLE WITH THE NEW ORDER INFORMATION.      */
/* FINALLY, INSERT THE NEW ORDER ITEMS AND DENOMINATIONS.            */
/*********************************************************************/

CALL INIT_DATA;

CALL DELETE_OLD_ORDER_INFO;

IF PGM_RC = PGM_SUCCESS THEN
  DO;
    CALL UPD_ORDER;
    IF SQLCODE = SQL_GOOD THEN
      CALL PROCESS_NEW_ORDER_ITEMS;
    ELSE
      DO;
        PGM_RC = PGM_FAILURE;
        PGM_TEXT = 'BAD SQLCODE UPDATING ORDER ROW';
        EXEC CICS SYNCPOINT ROLLBACK;
      END;
  END;
ELSE
  EXEC CICS SYNCPOINT ROLLBACK;

DB2_SQLCODE = SQLCODE;
EXEC SQL COMMIT;  /************************************** ITSO */

EXEC CICS SYNCPOINT;

EXEC CICS RETURN;

/*********************************************************************/
/* SUBROUTINE TO INITIALIZE PROGRAM DATA AREAS                       */
/*********************************************************************/

INIT_DATA: PROC;

  COMMAREA.PGM_NAME          = 'PNEWOR3';

  DCLORDER.ORDER_NUM         = O_NUM;
  DCLORDER_ITEM.ORDER_NUM    = O_NUM;
  DCLORDER_DENOM.ORDER_NUM   = O_NUM;

  DCLORDER.VALU_ORD_NEW_CURR = O_VALU_ORD_NEW_CURR;
  DCLORDER.PROFIT            = O_PROFIT;
  DCLORDER.TOTAL             = O_TOTAL;
  DCLORDER.ORDER_STATUS      = O_STATUS;
  DCLORDER.LAST_DATE_UPD     = O_LAST_DATE_UPD;
  DCLORDER.LAST_TIME_UPD     = O_LAST_TIME_UPD;
  DCLORDER.BANK_CODE         = B_CODE;
  DCLORDER.BANK_NAME         = B_NAME;
  DCLORDER.BANK_COMMISSION   = B_COMMISSION;
```

Figure 105 (Part 2 of 5). Listing of Program PNEWOR3

```
  DCLORDER.BRANCH_CODE       = BR_CODE;
  DCLORDER.BRANCH_NAME       = BR_NAME;
  DCLORDER.BRANCH_ADDRESS    = BR_ADDRESS;
  DCLORDER.CASHIER_ID        = C_ID;
  DCLORDER.CASHIER_NAME      = C_NAME;
  DCLORDER.CUST_ID           = CU_ID;
  DCLORDER.CUST_NAME         = CU_NAME;
  DCLORDER.CUST_ADDRESS      = CU_ADDRESS;

  EXEC SQL CONNECT TO SRVDB; /***************************** ITSO */

END INIT_DATA;

/*********************************************************************/
/* SUBROUTINE TO DELETE ALL OLD ORDER INFO, IF IT EXISTS             */
/*********************************************************************/

DELETE_OLD_ORDER_INFO: PROC;

  CALL DELETE_OLD_ORDER_ITEMS;
  IF SQLCODE = SQL_GOOD THEN
    DO;
      PGM_RC = PGM_SUCCESS;
      CALL DELETE_OLD_DEN;
      IF SQLCODE = SQL_GOOD THEN
        PGM_RC = PGM_SUCCESS;
      ELSE
        IF SQLCODE = SQL_NO_DATA THEN
          PGM_RC = PGM_SUCCESS;
        ELSE
          DO;
            PGM_RC = PGM_FAILURE;
            PGM_TEXT = 'BAD SQLCODE DELETING ORDER DEN DATA';
          END;
    END;
  ELSE
    IF SQLCODE = SQL_NO_DATA THEN
      PGM_RC = PGM_SUCCESS;
    ELSE
      DO;
        PGM_RC = PGM_FAILURE;
        PGM_TEXT = 'BAD SQLCODE DELETING ORDER ITEM DATA';
      END;

END DELETE_OLD_ORDER_INFO;

/*********************************************************************/
/* SUBROUTINE TO DELETE ALL ORDER ITEMS ASSOCIATED WITH ORDER        */
/*********************************************************************/

DELETE_OLD_ORDER_ITEMS: PROC;

  EXEC SQL DELETE FROM SRV.ORDER_ITEM
    WHERE  ORDER_NUM = :DCLORDER_ITEM.ORDER_NUM;

END DELETE_OLD_ORDER_ITEMS;

/*********************************************************************/
/* SUBROUTINE TO DELETE ALL ORDER DENOMIMATIONS FOR ORDER            */
/*********************************************************************/

DELETE_OLD_DEN: PROC;

  EXEC SQL DELETE FROM SRV.ORDER_DENOM
    WHERE  ORDER_NUM = :DCLORDER_DENOM.ORDER_NUM;

END DELETE_OLD_DEN;

/*********************************************************************/
/* SUBROUTINE TO UPDATE 'INITIALIZED' ORDER TO 'NEW' ORDER           */
/*********************************************************************/
```

Figure 105 (Part 3 of 5). Listing of Program PNEWOR3

```
UPD_ORDER: PROC;

  EXEC SQL UPDATE SRV.ORDERT                              /* changed */
  SET
    VALU_ORD_NEW_CURR = :DCLORDER.VALU_ORD_NEW_CURR ,
    PROFIT            = :DCLORDER.PROFIT            ,
    TOTAL             = :DCLORDER.TOTAL             ,
    ORDER_STATUS      = :DCLORDER.ORDER_STATUS      ,
    LAST_DATE_UPD     = :DCLORDER.LAST_DATE_UPD     ,
    LAST_TIME_UPD     = :DCLORDER.LAST_TIME_UPD     ,
    BANK_CODE         = :DCLORDER.BANK_CODE         ,
    BANK_NAME         = :DCLORDER.BANK_NAME         ,
    BANK_COMMISSION   = :DCLORDER.BANK_COMMISSION   ,
    BRANCH_CODE       = :DCLORDER.BRANCH_CODE       ,
    BRANCH_NAME       = :DCLORDER.BRANCH_NAME       ,
    BRANCH_ADDRESS    = :DCLORDER.BRANCH_ADDRESS    ,
    CASHIER_ID        = :DCLORDER.CASHIER_ID        ,
    CASHIER_NAME      = :DCLORDER.CASHIER_NAME      ,
    CUST_ID           = :DCLORDER.CUST_ID           ,
    CUST_NAME         = :DCLORDER.CUST_NAME         ,
    CUST_ADDRESS      = :DCLORDER.CUST_ADDRESS
  WHERE
    ORDER_NUM         = :DCLORDER.ORDER_NUM;

END UPD_ORDER;

/*********************************************************************/
/* SUBROUTINE TO PROCESS THE ORDER ITEMS WITHIN THE ORDER           */
/*********************************************************************/

PROCESS_NEW_ORDER_ITEMS: PROC;

  CNT_ITEM = 0;
  QUIT_ITEM_LOOP = FALSE_F;
  DO WHILE(QUIT_ITEM_LOOP = FALSE_F);
    CNT_ITEM = CNT_ITEM + 1;
    IF O_ITEM_NUM(CNT_ITEM) ¬= 0 THEN
      DO;
        CALL INSERT_ITEM;
        CNT_DEN = 0;
        QUIT_DEN_LOOP = FALSE_F;
        DO WHILE(QUIT_DEN_LOOP = FALSE_F);
          CNT_DEN = CNT_DEN + 1;
          IF O_DENOMINATION(CNT_ITEM,CNT_DEN) ¬= 0 THEN
            CALL INSERT_DEN;
          ELSE
            QUIT_DEN_LOOP = TRUE_T;
        END; /* INNER WHILE LOOP */
      END;
    ELSE
      QUIT_ITEM_LOOP = TRUE_T;
  END; /* OUTER WHILE LOOP */

  IF SQLCODE = SQL_GOOD THEN
    DO;
      PGM_RC   = PGM_SUCCESS;
      PGM_TEXT = 'SUCCESSFUL PROGRAM COMPLETION';
    END;
  ELSE
    DO;
      PGM_RC   = PGM_FAILURE;
      PGM_TEXT = 'BAD SQLCODE INSERTING ORDER ITEM DATA';
      EXEC CICS SYNCPOINT ROLLBACK;
    END;

END PROCESS_NEW_ORDER_ITEMS;

/*********************************************************************/
/* SUBROUTINE TO INSERT A ROW INTO THE SRV.ORDER_ITEM TABLE         */
/*********************************************************************/

INSERT_ITEM: PROC;
```

Figure 105 (Part 4 of 5). Listing of Program PNEWOR3

```
  DCLORDER_ITEM.ORDER_ITEM_NUM  = O_ITEM_NUM(CNT_ITEM);
  DCLORDER_DENOM.ORDER_ITEM_NUM = O_ITEM_NUM(CNT_ITEM);
  TYPE_OF_CURRENCY              = O_CURRENCY_TYPE(CNT_ITEM);
  XCHNG_RATE                    = O_XCHNG_RATE(CNT_ITEM);
  VALU_NEW_CURR                 = O_VALU_NEW_CURR(CNT_ITEM);
  VALU_ORIG_CURR                = O_VALU_ORIG_CURR(CNT_ITEM);

  EXEC SQL INSERT INTO SRV.ORDER_ITEM
   ( ORDER_NUM         ,
     ORDER_ITEM_NUM    ,
     TYPE_OF_CURRENCY  ,
     XCHNG_RATE        ,
     VALU_NEW_CURR     ,
     VALU_ORIG_CURR    )
  VALUES
   ( :DCLORDER_ITEM.ORDER_NUM        ,
     :DCLORDER_ITEM.ORDER_ITEM_NUM   ,
     :DCLORDER_ITEM.TYPE_OF_CURRENCY ,
     :DCLORDER_ITEM.XCHNG_RATE       ,
     :DCLORDER_ITEM.VALU_NEW_CURR    ,
     :DCLORDER_ITEM.VALU_ORIG_CURR   );

  IF SQLCODE = SQL_GOOD THEN
    IF CNT_ITEM = MAX_NUM_ITEM THEN
      QUIT_ITEM_LOOP = TRUE_T;
    ELSE;
  ELSE
    QUIT_ITEM_LOOP = TRUE_T;

END INSERT_ITEM;

/*********************************************************************/
/* SUBROUTINE TO INSERT A ROW INTO THE SRV.ORDER_DENOM TABLE        */
/*********************************************************************/

INSERT_DEN: PROC;

  DENOMINATION = O_DENOMINATION(CNT_ITEM,CNT_DEN);
  NUM_BILLS    = O_NUM_BILLS(CNT_ITEM,CNT_DEN);
  VALU         = O_VALU(CNT_ITEM,CNT_DEN);

  EXEC SQL INSERT INTO SRV.ORDER_DENOM
   ( ORDER_NUM       ,
     ORDER_ITEM_NUM  ,
     DENOMINATION    ,
     NUM_BILLS       ,
     VALU            )
  VALUES
   ( :DCLORDER_DENOM.ORDER_NUM      ,
     :DCLORDER_DENOM.ORDER_ITEM_NUM ,
     :DCLORDER_DENOM.DENOMINATION   ,
     :DCLORDER_DENOM.NUM_BILLS      ,
     :DCLORDER_DENOM.VALU           );

  IF SQLCODE = SQL_GOOD THEN
    IF CNT_DEN = MAX_NUM_DEN THEN
      QUIT_DEN_LOOP = TRUE_T;
    ELSE;
  ELSE
    DO;
      QUIT_DEN_LOOP = TRUE_T;
      QUIT_ITEM_LOOP = TRUE_T;
    END;

END INSERT_DEN;

END PNEWOR3;
```

Figure 105 (Part 5 of 5). Listing of Program PNEWOR3

Program PLISTUB

This CICS transaction program calls the two CICS DB2 programs (PCASH and PNEWOR3) for testing. It is invoked using the CICS TCLL transaction, and it writes output to a CICS temporary or transient data queue.

```
PLISTUB: PROC OPTIONS(MAIN REENTRANT);

    /*****************************************************************
     * ITSC - SAN JOSE   CLIENT/SERVER COMPUTING USING DCE/AS        *
     *                                                               *
     * ORIGINAL PROGRAM IN COBOL, REWRITTEN IN PL/I,                 *
     * EXTRACTED RELEVANT SECTIONS.                                  *
     *                                                               *
     * DESCRIPTION:  CALL USER-INPUTTED PROGRAM FOR TESTING PURPOSES *
     *                                                               *
     *               (lots of text...)                               *
     *****************************************************************

    /*****************************************************************
     * COMMAREAS USED TO PASS DATA BETWEEN CLIENT AND SERVER         *
     *****************************************************************/
    DCL
         DBUGIN             CHAR(8)    INIT('DBUGIN'),
         DBUGOUT            CHAR(8)    INIT('DBUGOUT'),
         PLIO               CHAR(4)    INIT('PLIO'),
         CICSRC             BIN FIXED(31),

      01 INPUT_AREA,
         05 CICS_TRX_CODE   CHAR(4)    INIT(' '),
         05 FILLER1         CHAR(1)    INIT(' '),
         05 PROG_ID         CHAR(8)    INIT(' '),
         05 FILLER2         CHAR(1)    INIT(' '),
         05 PROG_DATA       CHAR(100)  INIT(' '),
      01 INPUT_CICS         CHAR(110) DEFINED INPUT_AREA POS(1),
      01 INPUT_EDF          CHAR(110) DEFINED INPUT_AREA POS(4),

      01 DEBUG_COMMAREA     CHAR(2000) INIT(' '),
         COMMAREA_PTR       PTR        INIT(ADDR(DEBUG_COMMAREA)),
      01 DEBUG_OUTPUT       CHAR(132)  BASED(COMMAREA_PTR);

    DCL
      01 CASHIER_COM        BASED(COMMAREA_PTR) UNALIGNED,
         05 DB_DATA,
            %INCLUDE PCASHCOM;
         05 MESSAGES,
            %INCLUDE PMSGCOM;
    DCL
      01 ORDER_COM          BASED(COMMAREA_PTR) UNALIGNED,
         05 DB_DATA,
            %INCLUDE PORDCOM;
         05 MESSAGES,
            %INCLUDE PMSGCOM;

    /*****************************************************************
     * PROCEDURE DIVISION CONTAINING MAINLINE PROGRAM LOGIC          *
     *****************************************************************/

          CALL INIT_DATA;

          CALL LINK_TO_PROG;

          CALL WRITE_OUT_DATA;

          EXEC CICS SEND CONTROL FREEKB ERASE;
          EXEC CICS RETURN;

    /*****************************************************************
     * PERFORM ROUTINE TO INITIALIZE PROGRAM DATA AREAS              *
     * READ THE USER ENTERED INPUT TO SEE WHICH PROGRAM TO LINK TO.  *
```

Figure 106 (Part 1 of 3). Listing of Program PLISTUB

```
     ****************************************************************/
INIT_DATA:  PROC;
          EXEC CICS RECEIVE INTO(INPUT_AREA);
          IF CICS_TRX_CODE ¬= 'TCLL'  &         /* necessary on OS/2 */
             SUBSTR(INPUT_EDF,1,4) = 'TCLL'  THEN
                INPUT_CICS = INPUT_EDF;
          IF PROG_DATA ¬= ' ' THEN  DEBUG_COMMAREA = PROG_DATA;
                              ELSE  CALL SET_DEFAULT_DATA;
END INIT_DATA;

    /*****************************************************************
     * PERFORM ROUTINE TO SET DEFAULT DATA IN COMMAREA               *
     *****************************************************************/
SET_DEFAULT_DATA:  PROC;
          IF PROG_ID = ' ' THEN EXEC CICS RETURN;

    /**** INPUT TO PCASHIER (PLI VERSION) PROGRAM IS CASHIER ID */
          IF PROG_ID = 'PCASH' THEN
             CASHIER_COM.DB_DATA.C_ID = '1234';

    /**** INPUT TO PNEWOR3  PROGRAM IS THE FULL ORDER INFORMATION */
          IF PROG_ID = 'PNEWOR3'  THEN DO;
             ORDER_COM.DB_DATA.O_NUM             = 999;
             ORDER_COM.DB_DATA.O_VALU_ORD_NEW_CURR = '999.99';
             ORDER_COM.DB_DATA.O_PROFIT          = '99.9';
             ORDER_COM.DB_DATA.O_TOTAL           = '99999.99';
             ORDER_COM.DB_DATA.O_STATUS          = 'COM';
             ORDER_COM.DB_DATA.O_LAST_DATE_UPD   = '1994-10-24';
             ORDER_COM.DB_DATA.O_LAST_TIME_UPD   = '12.12.12';
             ORDER_COM.DB_DATA.B_COMMISSION      = '0.022';
             ORDER_COM.DB_DATA.B_CODE            = 'BANKCODE';
             ORDER_COM.DB_DATA.B_NAME            = 'BANK NAME';
             ORDER_COM.DB_DATA.BR_CODE           = 'BRNCHCDE';
             ORDER_COM.DB_DATA.BR_NAME           = 'BRANCH NAME';
             ORDER_COM.DB_DATA.BR_ADDRESS        = 'BRNACH ADDRESS';
             ORDER_COM.DB_DATA.C_ID              = '12345678';
             ORDER_COM.DB_DATA.C_NAME            = 'MIKE MAROUPD';
             ORDER_COM.DB_DATA.CU_ID             = 'CUSTID';
             ORDER_COM.DB_DATA.CU_NAME           = 'CUSTOMER NAME';
             ORDER_COM.DB_DATA.CU_ADDRESS        = 'CUSTOMER ADDRESS';

             ORDER_COM.DB_DATA.O_ITEM_NUM(1)       = 1;
             ORDER_COM.DB_DATA.O_CURRENCY_TYPE(1)  = 'USD';
             ORDER_COM.DB_DATA.O_XCHNG_RATE(1)     = 0.0222;
             ORDER_COM.DB_DATA.O_VALU_NEW_CURR(1)  = 222222;
             ORDER_COM.DB_DATA.O_VALU_ORIG_CURR(1) = '2222222222';
             ORDER_COM.DB_DATA.O_DENOMINATION(1,1) = 20;
             ORDER_COM.DB_DATA.O_NUM_BILLS(1,1)    = 2;
             ORDER_COM.DB_DATA.O_VALU(1,1)         = 40;
             ORDER_COM.DB_DATA.O_DENOMINATION(1,2) = 50;
             ORDER_COM.DB_DATA.O_NUM_BILLS(1,2)    = 1;
             ORDER_COM.DB_DATA.O_VALU(1,2)         = 50;
             ORDER_COM.DB_DATA.O_DENOMINATION(1,3) = 0;
             ORDER_COM.DB_DATA.O_NUM_BILLS(1,3)    = 0;
             ORDER_COM.DB_DATA.O_VALU(1,3)         = 0;

             ORDER_COM.DB_DATA.O_ITEM_NUM(2)       = 0;
             ORDER_COM.DB_DATA.O_CURRENCY_TYPE(2)  = ' ';
             ORDER_COM.DB_DATA.O_XCHNG_RATE(2)     = 0;
             ORDER_COM.DB_DATA.O_VALU_NEW_CURR(2)  = 0;
             ORDER_COM.DB_DATA.O_VALU_ORIG_CURR(2) = ' ';
          END;
END SET_DEFAULT_DATA;

    /*****************************************************************
     * PERFORM ROUTINE TO RETRIEVE DATA INTO DFHCOMMAREA             *
     *****************************************************************/
LINK_TO_PROG:  PROC;
          EXEC CICS LINK PROGRAM(PROG_ID) COMMAREA(DEBUG_COMMAREA);
END LINK_TO_PROG;

WRITE_OUT_DATA: PROC;
```

Figure 106 (Part 2 of 3). Listing of Program PLISTUB

```
       /* EXEC CICS WRITEQ TS QUEUE(DBUGOUT) FROM(DEBUG_COMMAREA); */
          EXEC CICS WRITEQ TD QUEUE(PLIO) FROM(DEBUG_OUTPUT);
END WRITE_OUT_DATA;

END PLISTUB;
```

Figure 106 (Part 3 of 3). Listing of Program PLISTUB

Program PLIECI

This program uses the CICS External Call Interface (ECI) to invoke the two CICS DB2 programs (PCASH and PNEWOR3) from outside CICS.

```
*process langlvl(saa2);
 /*********************************************************************/
 /* MODULE NAME        PLIECI.PLI                                     */
 /*                                                                   */
 /* DESCRIPTIVE NAME   Sample ECI program                             */
 /*                    Call other PLI CICS transactions               */
 /*********************************************************************/
 PLIECI:  proc options(main) returns(fixed bin(15));

 /* ECI include file: */
 %include cics_eci;

 DCL
       eci_parms_s           type ECI_PARMS,
       sEciReturnCode        fixed bin(15),
       DSrc                  fixed bin(31);
 DCL
       01 COMMAREA           CHAR(2000) INIT(' '),
          COMMAREA_PTR       PTR        INIT(ADDR(COMMAREA)),
       01 OUTPUT             CHAR(132)  BASED(COMMAREA_PTR);
 DCL
       01 CASHIER_COM        BASED(COMMAREA_PTR) UNALIGNED,
          05 DB_DATA,
             %INCLUDE PCASHCOM;
          05 MESSAGES,
             %INCLUDE PMSGCOM;
 DCL
       01 ORDER_COM          BASED(COMMAREA_PTR) UNALIGNED,
          05 DB_DATA,
             %INCLUDE PORDCOM;
          05 MESSAGES,
             %INCLUDE PMSGCOM;

    /*****************************************************************/
    /* Tell the user that we're starting:                            */
    /*****************************************************************/
    display ( " " );
    display ( "PLIECI: starting" );

    /*****************************************************************/
    /* Initialize the ECI structures:                                */
    /*****************************************************************/
    unspec(eci_parms_s)  = ''b;
    eci_parms_s.eci_call_type       = ECI_SYNC;
    eci_parms_s.eci_userid          = "SYSAD";
    eci_parms_s.eci_password        = "SYSAD";
    eci_parms_s.eci_commarea        = addr(COMMAREA);
    eci_parms_s.eci_commarea_length = stg(COMMAREA);
    eci_parms_s.eci_timeout         = 60;

    /*****************************************************************/
    /* Tell the user that we're calling ECI now                      */
```

Figure 107 (Part 1 of 3). Listing of Program PLIECI

```
/*****************************************************************/
display ( "PLIECI: Preparing a call to PCASH" );

eci_parms_s.eci_program_name    = "PCASH";
       CASHIER_COM.DB_DATA.C_ID = '1234';

sEciReturnCode = FaaExternalCall (eci_parms_s);         /* ECI Call */

display ( " - return code: " || sEciReturnCode);
display ( " - cash id    : " || cashier_com.db_data.c_id);
display ( " -      name  : " || cashier_com.db_data.c_name);
display ( " -      code  : " || cashier_com.db_data.b_code);
display ( " - db2  rc    : " || cashier_com.messages.db2_sqlcode);
display ( " - pgm  rc    : " || cashier_com.messages.pgm_rc);
display ( " -      name  : " || cashier_com.messages.pgm_name);
display ( " -      text  : " || cashier_com.messages.pgm_text);

display ( "PLIECI: Preparing a call to PNEWOR3" );

eci_parms_s.eci_program_name    = "PNEWOR3";
          ORDER_COM.DB_DATA.O_NUM             = 999;
          ORDER_COM.DB_DATA.O_VALU_ORD_NEW_CURR = '999.99';
          ORDER_COM.DB_DATA.O_PROFIT          = '99.9';
          ORDER_COM.DB_DATA.O_TOTAL           = '99999.99';
          ORDER_COM.DB_DATA.O_STATUS          = 'COM';
          ORDER_COM.DB_DATA.O_LAST_DATE_UPD   = '1994-10-24';
          ORDER_COM.DB_DATA.O_LAST_TIME_UPD   = '12.12.12';
          ORDER_COM.DB_DATA.B_COMMISSION      = '0.022';
          ORDER_COM.DB_DATA.B_CODE            = 'BANKCODE';
          ORDER_COM.DB_DATA.B_NAME            = 'BANK NAME';
          ORDER_COM.DB_DATA.BR_CODE           = 'BRNCHCDE';
          ORDER_COM.DB_DATA.BR_NAME           = 'BRANCH NAME';
          ORDER_COM.DB_DATA.BR_ADDRESS        = 'BRNACH ADDRESS';
          ORDER_COM.DB_DATA.C_ID              = '12345678';
          ORDER_COM.DB_DATA.C_NAME            = 'MIKE MAROUPD';
          ORDER_COM.DB_DATA.CU_ID             = 'CUSTID';
          ORDER_COM.DB_DATA.CU_NAME           = 'CUSTOMER NAME';
          ORDER_COM.DB_DATA.CU_ADDRESS        = 'CUSTOMER ADDRESS';

          ORDER_COM.DB_DATA.O_ITEM_NUM(1)       = 1;
          ORDER_COM.DB_DATA.O_CURRENCY_TYPE(1)  = 'USD';
          ORDER_COM.DB_DATA.O_XCHNG_RATE(1)     = 0.0222;
          ORDER_COM.DB_DATA.O_VALU_NEW_CURR(1)  = 222222;
          ORDER_COM.DB_DATA.O_VALU_ORIG_CURR(1) = '2222222222';

          ORDER_COM.DB_DATA.O_DENOMINATION(1,1) = 20;
          ORDER_COM.DB_DATA.O_NUM_BILLS(1,1)    = 2;
          ORDER_COM.DB_DATA.O_VALU(1,1)         = 40;
          ORDER_COM.DB_DATA.O_DENOMINATION(1,2) = 50;
          ORDER_COM.DB_DATA.O_NUM_BILLS(1,2)    = 1;
          ORDER_COM.DB_DATA.O_VALU(1,2)         = 50;
          ORDER_COM.DB_DATA.O_DENOMINATION(1,3) = 0;
          ORDER_COM.DB_DATA.O_NUM_BILLS(1,3)    = 0;
          ORDER_COM.DB_DATA.O_VALU(1,3)         = 0;

          ORDER_COM.DB_DATA.O_ITEM_NUM(2)       = 0;
          ORDER_COM.DB_DATA.O_CURRENCY_TYPE(2)  = ' ';
          ORDER_COM.DB_DATA.O_XCHNG_RATE(2)     = 0;
          ORDER_COM.DB_DATA.O_VALU_NEW_CURR(2)  = 0;
          ORDER_COM.DB_DATA.O_VALU_ORIG_CURR(2) = ' ';

sEciReturnCode = FaaExternalCall (eci_parms_s);         /* ECI Call */

display ( " - return code: " || sEciReturnCode);
display ( " - db2  rc    : " || order_com.messages.db2_sqlcode);
display ( " - pgm  rc    : " || order_com.messages.pgm_rc);
display ( " -      name  : " || order_com.messages.pgm_name);
display ( " -      text  : " || order_com.messages.pgm_text);

/*****************************************************************/
/* Return 0 to show that this program has succeeded:             */
/*****************************************************************/
```

Figure 107 (Part 2 of 3). Listing of Program PLIECI

```
    display ( "PLIECI: end" );
    return (0);

END PLIECI;
```

Figure 107 (Part 3 of 3). Listing of Program PLIECI

Include Members

Several include members are used to build the communications area and host structures for the DB2 tables.

```
Member   Code
-------- ---------------------------------------------------------------------

PCOMITMS
         /* PLI VERSION OF COMMON PROGRAM ITEMS USED BY ALL SERVER PROGRAMS  */
         DCL SQL_GOOD      FIXED BIN(31,0) INIT(0);
         DCL SQL_NO_DATA   FIXED BIN(31,0) INIT(100);
         DCL SQL_DUPL_ROW  FIXED BIN(31,0) INIT(-803);
         DCL PGM_SUCCESS   CHAR(01) INIT('S');
         DCL PGM_NO_DATA   CHAR(01) INIT('N');
         DCL PGM_FAILURE   CHAR(01) INIT('F');

PMSGCOM
         /* PLI VERSION OF COMMAREA STRUCTURE FOR MESSAGE AREA              */
                 1O DB2_SQLCODE  FIXED BIN(31,0),
                 1O PGM_RC       CHAR(01),
                 1O PGM_NAME     CHAR(08),
                 1O PGM_TEXT     CHAR(50);

PCASHCOM
         /* PLI VERSION OF COMMAREA STRUCTURE FOR CASHIER DATA              */
                 1O C_ID         CHAR(08),
                 1O C_NAME       CHAR(40),
                 1O B_CODE       CHAR(08),

PORDCOM
         /* PLI VERSION OF COMMAREA STRUCTURE FOR ORDER DETAILS             */
         /*      GENERAL INFORMATION FOR ORDER                              */
                 1O O_NUM                FIXED BIN(31,0),
                 1O O_VALU_ORD_NEW_CURR  CHAR(20),
                 1O O_PROFIT             CHAR(20),
                 1O O_TOTAL              CHAR(20),
                 1O O_STATUS             CHAR(3),
                 1O O_LAST_DATE_UPD      CHAR(10),
                 1O O_LAST_TIME_UPD      CHAR(8),
         /*      BANK INFORMATION FOR ORDER                                 */
                 1O B_CODE               CHAR(8),
                 1O B_NAME               CHAR(40),
                 1O B_COMMISSION         DEC FLOAT(4),
         /*      BRANCH INFORMATION FOR ORDER                               */
                 1O BR_CODE              CHAR(8),
                 1O BR_NAME              CHAR(40),
                 1O BR_ADDRESS           CHAR(80),
         /*      CASHIER INFORMATION FOR ORDER                              */
                 1O C_ID                 CHAR(8),
                 1O C_NAME               CHAR(40),
         /*      CUSTOMER INFORMATION FOR ORDER                             */
                 1O CU_ID                CHAR(6),
                 1O CU_NAME              CHAR(40),
                 1O CU_ADDRESS           CHAR(80),
         /*      THERE CAN BE UP TO 1O ORDER ITEMS FOR EACH ORDER           */
                 1O ORDER_ITEM_DETAILS (10),
```

Figure 108 (Part 1 of 3). Listing of Include Members

```
                15 O_ITEM_NUM           FIXED BIN(15,0),
                15 O_CURRENCY_TYPE      CHAR(3),
                15 O_XCHNG_RATE         FIXED BIN(31,0),
                15 O_VALU_NEW_CURR      FIXED BIN(31,0),
                15 O_VALU_ORIG_CURR     CHAR(20),
        /*      EACH ORDER ITEM CAN HAVE UP TO 9 DIFFERENT DENOMINATIONS  */
                15 DENOM_DETAILS (9),
                  20 O_DENOMINATION     FIXED BIN(31,0),
                  20 O_NUM_BILLS        FIXED BIN(15,0),
                  20 O_VALU             FIXED BIN(31,0),

PCASHIER
        /************************************************************/
        /* DCLGEN COMMAND MADE THE FOLLOWING STATEMENTS             */
        /************************************************************/
            EXEC SQL DECLARE SRV.CASHIER TABLE
            (
             CASHIER_NAME                   CHAR(40) NOT NULL ,
             CASHIER_ID                     CHAR(8) NOT NULL ,
             BRANCH_CODE                    CHAR(8) NOT NULL
            ) ;
        /************************************************************/
        /* PL/I DECLARATION FOR TABLE CASHIER                       */
        /************************************************************/
        DCL 1 DCLCASHIER,
             5 CASHIER_NAME                   CHAR(40),
             5 CASHIER_ID                     CHAR(8),
             5 BRANCH_CODE                    CHAR(8);
        /************************************************************/
        /* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 3   */
        /************************************************************/

PORDER
        /************************************************************/
        /* DCLGEN COMMAND MADE THE FOLLOWING STATEMENTS             */
        /************************************************************/
            EXEC SQL DECLARE SRV.ORDERT TABLE
            (
             ORDER_NUM                      INTEGER NOT NULL ,
             VALU_ORD_NEW_CURR              CHAR(20) ,
             PROFIT                         CHAR(20) ,
             TOTAL                          CHAR(20) ,
             ORDER_STATUS                   CHAR(3) NOT NULL ,
             LAST_DATE_UPD                  DATE NOT NULL ,
             LAST_TIME_UPD                  TIME NOT NULL ,
             BANK_CODE                      CHAR(8) NOT NULL ,
             BANK_NAME                      CHAR(40) NOT NULL ,
             BANK_COMMISSION                DECIMAL(4,3) NOT NULL ,
             BRANCH_CODE                    CHAR(8) NOT NULL ,
             BRANCH_NAME                    CHAR(40) NOT NULL ,
             BRANCH_ADDRESS                 CHAR(80) NOT NULL ,
             CASHIER_ID                     CHAR(8) NOT NULL ,
             CASHIER_NAME                   CHAR(40) NOT NULL ,
             CUST_ID                        CHAR(6) ,
             CUST_NAME                      CHAR(40) ,
             CUST_ADDRESS                   CHAR(80)
            ) ;
        /************************************************************/
        /* PL/I DECLARATION FOR TABLE ORDERT                        */
        /************************************************************/
        DCL 1 DCLORDER,
             5 ORDER_NUM                      FIXED BIN (31),
             5 VALU_ORD_NEW_CURR              CHAR(20),
             5 PROFIT                         CHAR(20),
             5 TOTAL                          CHAR(20),
             5 ORDER_STATUS                   CHAR(3),
             5 LAST_DATE_UPD                  CHAR(10),
             5 LAST_TIME_UPD                  CHAR(8),
             5 BANK_CODE                      CHAR(8),
             5 BANK_NAME                      CHAR(40),
             5 BANK_COMMISSION                FIXED DECIMAL(4,3),
             5 BRANCH_CODE                    CHAR(8),
```

Figure 108 (Part 2 of 3). Listing of Include Members

```
          5 BRANCH_NAME                    CHAR(40),
          5 BRANCH_ADDRESS                 CHAR(80),
          5 CASHIER_ID                     CHAR(8),
          5 CASHIER_NAME                   CHAR(40),
          5 CUST_ID                        CHAR(6),
          5 CUST_NAME                      CHAR(40),
          5 CUST_ADDRESS                   CHAR(80);
        /************************************************************/
        /* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 18  */
        /************************************************************/

PORDITM
        /************************************************************/
        /* DCLGEN COMMAND MADE THE FOLLOWING STATEMENTS               */
        /************************************************************/
            EXEC SQL DECLARE SRV.ORDER_ITEM TABLE
            (
             ORDER_NUM                     INTEGER NOT NULL ,
             ORDER_ITEM_NUM                SMALL NOT NULL ,
             TYPE_OF_CURRENCY              CHAR(3) NOT NULL ,
             XCHNG_RATE                    FLOAT(53) NOT NULL ,
             VALU_NEW_CURR                 INTEGER NOT NULL ,
             VALU_ORIG_CURR                CHAR(20) NOT NULL
            ) ;
        /************************************************************/
        /* PL/I DECLARATION FOR TABLE ORDER_ITEM                      */
        /************************************************************/
        DCL 1 DCLORDER_ITEM,
             5 ORDER_NUM                     FIXED BIN (31),
             5 ORDER_ITEM_NUM                FIXED BIN(15),
             5 TYPE_OF_CURRENCY              CHAR(3),
             5 XCHNG_RATE                    FLOAT BIN (53),
             5 VALU_NEW_CURR                 FIXED BIN (31),
             5 VALU_ORIG_CURR                CHAR(20);
        /************************************************************/
        /* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 6   */
        /************************************************************/

PORDEN
        /************************************************************/
        /* DCLGEN COMMAND MADE THE FOLLOWING STATEMENTS               */
        /************************************************************/
            EXEC SQL DECLARE SRV.ORDER_DENOM TABLE
            (
             ORDER_NUM                     INTEGER NOT NULL ,
             ORDER_ITEM_NUM                SMALL NOT NULL ,
             DENOMINATION                  INTEGER NOT NULL ,
             NUM_BILLS                     SMALL NOT NULL ,
             VALU                          INTEGER NOT NULL
            ) ;
        /************************************************************/
        /* PL/I DECLARATION FOR TABLE ORDER_DENOM                     */
        /************************************************************/
        DCL 1 DCLORDER_DENOM,
             5 ORDER_NUM                     FIXED BIN (31),
             5 ORDER_ITEM_NUM                FIXED BIN(15),
             5 DENOMINATION                  FIXED BIN (31),
             5 NUM_BILLS                     FIXED BIN(15),
             5 VALU                          FIXED BIN (31);
        /************************************************************/
        /* THE NUMBER OF COLUMNS DESCRIBED BY THIS DECLARATION IS 5   */
        /************************************************************/
```

Figure 108 (Part 3 of 3). Listing of Include Members

Sample Code for Visual PL/I

Two code block libraries for Visual PL/I applications are presented below.

My Code Block Library

This sample code block library contains a few code blocks developed for the CICS preprocessor and debugging of Visual PL/I applications.

```
&&pmgmycb.obj BEGIN CICS Preprocessor
*process pp ( cics('deck edf') macro )

END
&&pmgmycb.obj BEGIN Check for Numeric
if WC_ENTRYFIELD VARNAME = '' |
   verify (WC_ENTRYFIELD VARNAME, ' 0123456789') ¬= 0 then

END
&&pmgmycb.obj BEGIN Display Procname/Sourceline
display ( procname() || sourceline );

END
&&pmgmycb.obj BEGIN Put Data (variable)
/* Put data (variable) code block */
put data ( VARIABLE "name of variable to display" );

END
&&pmgmycb.obj BEGIN Display (variable)
/* Display (variable) code block */
display ( VARIABLE "name of variable to display" ) ;

END
&&pmgmycb.obj BEGIN Call PLIDUMP
/* Call PLIDUMP code block */
Call PLIDUMP ('TFC','VARIABLE "enter dump title"') ;

END
&&pmgmycb.obj BEGIN Display Proc/Sourceline
/* Display procedure name and source line code block */
Display ( procname() || sourceline() ) ;

END
&&pmgmycb.obj BEGIN Display (text and variable)
/* Display a text and a variable code block */
Display ('VARIABLE "text to be displayed" ' || VARIABLE "variable name" ) ;

END
```

Figure 109. Sample PL/I Code Blocks

Visual PL/I Code Blocks for Device Independence

The INDEVPLI code block files contains code blocks to deal with different device types for Visual PL/I applications.

Note: This code has not been formally tested by IBM. You may use it at your own risk.

Content of Device Independent Code Block Library

The various code blocks in the INDEVPLI library are:

- Initialization
  - Global variables for Screen Sizes
  - Set Development Device
- Window creation and sizing (use the following code blocks in place of those from Visual PL/I)
  - Set Window Position
  - Create a Child Window
  - Create a Standard Window on DESKTOP
  - Create a Window from a DLL
- Control creation and sizing
  - Set Listbox Position
  - Set MLE Position
  - Set Push Button Position
  - Set Entry Field Position
  - Set Static Text Position

Device Independent Code Block Library INDEVPLI.PL

```
Module    indevpli.plm
Functions indevpli.plf
Constants indevpli.plc
```

Figure 110. PL/I Code Block Library for Device Independence

Device Independent Code Block Library INDEVPLI.PLM

```
Device Independent Functions
 &&indevpli.obj
```

Figure 111. PL/I Code Block Module for Device Independence

Device Independent Code Block Library INDEVPLI.PLC

```
#define VGA_Device 640
#define VGA_DeviceY 480
#define XGA_Device 1024
#define XGA_DeviceY 768
```

Figure 112. PL/I Constants for Device Independence

Device Independent Code Block Library INDEVPLI.PLF

```
&&indevpli.obj BEGIN # Global variables for Independent Device
/* VGA and XGA X and Y sizes */
GLOBAL_VARIABLES dcl VGA_Device  fixed bin(31) value(640);
GLOBAL_VARIABLES dcl VGA_DeviceY fixed bin(31) value(480);
GLOBAL_VARIABLES dcl XGA_Device  fixed bin(31) value(1024);
GLOBAL_VARIABLES dcl XGA_DeviceY fixed bin(31) value(768);
GLOBAL_VARIABLES dcl X_ScreenSZ  USHORT ;
GLOBAL_VARIABLES dcl Y_ScreenSZ  USHORT ;
GLOBAL_VARIABLES dcl X_DevelopSZ USHORT ;
GLOBAL_VARIABLES dcl Y_DevelopSZ USHORT ;
END

&&indevpli.obj BEGIN DEV: Set Development Device
                          /* Identify Development Display Device the PM Windows were */
                          /* Created under (XGA or VGA) */
PMGPROMPT "Select Current Development Device"
X_DevelopSZ = SELECT (VGA_Device,XGA_Device);
if X_DevelopSZ = XGA_Device then
     Y_DevelopSZ = XGA_DeviceY;
else Y_DevelopSZ = VGA_DeviceY;
                         /* Get display device the application is now running under */
X_ScreenSZ = WinQuerySysValue(HWND_DESKTOP,SV_CXSCREEN);
Y_ScreenSZ = WinQuerySysValue(HWND_DESKTOP,SV_CYSCREEN);
END

&&indevpli.obj BEGIN DEV: Set Window Position
                        /* Scale to current display Device: Set the window position */
                        /* Formula: Multiply Size Screen was created under by the
                           size the size of the current display device. Now divide the
                           result by the size of the development display device */
Call WinSetWindowPos(WINDOW HANDLE ,
                HWND_BOTTOM,
                floor((WINDOW XPOS * X_ScreenSZ) / X_DevelopSZ),
                floor((WINDOW YPOS * Y_ScreenSZ) / Y_DevelopSZ),
                floor((WINDOW XSIZE * X_ScreenSZ) / X_DevelopSZ),
                floor((WINDOW YSIZE * Y_ScreenSZ) / Y_DevelopSZ),
                ior( SWP_MOVE , SWP_SIZE , SWP_ACTIVATE ) ) ;
END

&&indevpli.obj BEGIN DEV: Set Listbox Position
                        /* Scale to current display Device: Set the ListBox position */
Call WinSetWindowPos(WinWindowFromID (hwnd0, WC_LISTBOX ID),
                HWND_BOTTOM,
                floor((WC_LISTBOX XPOS * X_ScreenSZ) / X_DevelopSZ),
                floor((WC_LISTBOX YPOS * Y_ScreenSZ) / Y_DevelopSZ),
                floor((WC_LISTBOX XSIZE * X_ScreenSZ) / X_DevelopSZ),
                floor((WC_LISTBOX YSIZE * Y_ScreenSZ) / Y_DevelopSZ),
                ior( SWP_MOVE , SWP_SIZE , SWP_ACTIVATE ) ) ;
END

&&indevpli.obj BEGIN DEV: Set MLE Position
                        /* Scale to current display Device: Set the MLE position */
Call WinSetWindowPos(WinWindowFromID (hwnd0, WC_MLE ID),
                HWND_BOTTOM,
                floor((WC_MLE XPOS * X_ScreenSZ) / X_DevelopSZ),
                floor((WC_MLE YPOS * Y_ScreenSZ) / Y_DevelopSZ),
                floor((WC_MLE XSIZE * X_ScreenSZ) / X_DevelopSZ),
                floor((WC_MLE YSIZE * Y_ScreenSZ) / Y_DevelopSZ),
                ior( SWP_MOVE , SWP_SIZE , SWP_ACTIVATE ) ) ;
END

&&indevpli.obj BEGIN DEV: Set Push Button Position
                        /* Scale to current display Device: Set the push button position */
Call WinSetWindowPos(WinWindowFromID (hwnd0, WC_BUTTON ID),
                HWND_BOTTOM,
```

Figure 113 (Part 1 of 3). PL/I Code Blocks for Device Independence

```
                floor((WC_BUTTON XPOS * X_ScreenSZ) / X_DevelopSZ),
                floor((WC_BUTTON YPOS * Y_ScreenSZ) / Y_DevelopSZ),
                floor((WC_BUTTON XSIZE * X_ScreenSZ) / X_DevelopSZ),
                floor((WC_BUTTON YSIZE * Y_ScreenSZ) / Y_DevelopSZ),
                ior( SWP_MOVE , SWP_SIZE , SWP_ACTIVATE ) ) ;
END

&&indevpli.obj BEGIN DEV: Set Entry Field Position
                      /* Scale to current display Device: Set the entry field position */
Call WinSetWindowPos(WinWindowFromID (hwnd0, WC_ENTRYFIELD ID),
                HWND_BOTTOM,
                floor((WC_ENTRYFIELD XPOS * X_ScreenSZ) / X_DevelopSZ),
                floor((WC_ENTRYFIELD YPOS * Y_ScreenSZ) / Y_DevelopSZ),
                floor((WC_ENTRYFIELD XSIZE * X_ScreenSZ) / X_DevelopSZ),
                floor((WC_ENTRYFIELD YSIZE * Y_ScreenSZ) / Y_DevelopSZ),
                ior( SWP_MOVE , SWP_SIZE , SWP_ACTIVATE ) ) ;
END

&&indevpli.obj BEGIN DEV: Set Static Text Position
                      /* Scale to current display Device: Set the text position */
Call WinSetWindowPos(WinWindowFromID (hwnd0, WC_STATIC ID),
                HWND_BOTTOM,
                floor((WC_STATIC XPOS * X_ScreenSZ) / X_DevelopSZ),
                floor((WC_STATIC YPOS * Y_ScreenSZ) / Y_DevelopSZ),
                floor((WC_STATIC XSIZE * X_ScreenSZ) / X_DevelopSZ),
                floor((WC_STATIC YSIZE * Y_ScreenSZ) / Y_DevelopSZ),
                ior( SWP_MOVE , SWP_SIZE , SWP_ACTIVATE ) ) ;
END

&&indevpli.obj BEGIN DEV: Create a child window
GLOBAL_VARIABLES HMODULE CHILD MODULEHANDLE;
INITFUNCTION CHILD MODULEHANDLE=hModule;
                     /* Create a child window with handle CHILD HANDLE */
                     /* and parent CHILD PARENT */
CHILD HANDLE = WinCreateStdWindow(CHILD PARENT,
                                  WS_VISIBLE,
                                  /* &CHILD FCFVAR, */
                                  addr(CHILD FCFVAR),
                                  CHILD CLASS,
                                  CHILD TITLE,
                                  null(),
                                  CHILD MODULEHANDLE,
                                  CHILD ID,
                                  /* (PHWND) & CHILD CLHANDLE) */
                                  addr(CHILD CLHANDLE));
                      /* Scale to current display Device: Set the child window position */
Call WinSetWindowPos(CHILD HANDLE,
                HWND_BOTTOM,
                floor((CHILD XPOS * X_ScreenSZ) / X_DevelopSZ),
                floor((CHILD YPOS * Y_ScreenSZ) / Y_DevelopSZ),
                floor((CHILD XSIZE * X_ScreenSZ) / X_DevelopSZ),
                floor((CHILD YSIZE * Y_ScreenSZ) / Y_DevelopSZ),
                ior( SWP_MOVE , SWP_SIZE , SWP_ACTIVATE ) ) ;
END

&&indevpli.obj BEGIN DEV: Create a window - DESKTOP
GLOBAL_VARIABLES dcl CHILD MODULEHANDLE HMODULE ;
INITFUNCTION CHILD MODULEHANDLE=hModule; /* QQQ case */
                     /* Create a secondary window with handle CHILD HANDLE */
                     /* and parent HWND_DESKTOP */
CHILD HANDLE = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  /* &CHILD FCFVAR, */
                                  addr(CHILD FCFVAR),
                                  CHILD CLASS,
                                  CHILD TITLE,
                                  null(),
                                  CHILD MODULEHANDLE,
                                  CHILD ID,
                                  /* (PHWND) & CHILD CLHANDLE) */
```

Figure 113 (Part 2 of 3). PL/I Code Blocks for Device Independence

```
                                addr(CHILD CLHANDLE));
                          /* Scale to current display Device: Set the window position */
Call WinSetWindowPos(CHILD HANDLE,
                HWND_BOTTOM,
                floor((CHILD XPOS * X_ScreenSZ) / X_DevelopSZ),
                floor((CHILD YPOS * Y_ScreenSZ) / Y_DevelopSZ),
                floor((CHILD XSIZE * X_ScreenSZ) / X_DevelopSZ),
                floor((CHILD YSIZE * Y_ScreenSZ) / Y_DevelopSZ),
                ior( SWP_MOVE , SWP_SIZE , SWP_ACTIVATE ) ) ;
END

&&indevpli.obj BEGIN DEV: Create a window
GLOBAL_VARIABLES HMODULE CHILD MODULEHANDLE;
INITFUNCTION CHILD MODULEHANDLE=hModule;
                          /* Create a secondary window with handle CHILD HANDLE */
                          /* and parent HWND_DESKTOP */
CHILD HANDLE = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &CHILD FCFVAR,
                                  CHILD CLASS,
                                  CHILD TITLE,
                                  null(),
                                  CHILD MODULEHANDLE,
                                  CHILD ID,
                                  (PHWND) & CHILD CLHANDLE);
                          /* Scale to current display Device: Set the window position */
Call WinSetWindowPos(CHILD HANDLE,
                HWND_BOTTOM,
                floor((CHILD XPOS * X_ScreenSZ) / X_DevelopSZ),
                floor((CHILD YPOS * Y_ScreenSZ) / Y_DevelopSZ),
                floor((CHILD XSIZE * X_ScreenSZ) / X_DevelopSZ),
                floor((CHILD YSIZE * Y_ScreenSZ) / Y_DevelopSZ),
                ior( SWP_MOVE , SWP_SIZE , SWP_ACTIVATE ) ) ;
                     /* Owner is primary window */
WinSetOwner(CHILD HANDLE,
            hwnd0);
END

&&indevpli.obj BEGIN DEV: Create a Window from a DLL
PROTOTYPES MRESULT EXPENTRY DLLWINDOW PROCEDURE( HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2);
GLOBAL_VARIABLES HMODULE DLLWINDOW MODULEHANDLE;
GLOBAL_VARIABLES ULONG DLLWINDOW FCFVAR;
GLOBAL_VARIABLES HWND DLLWINDOW HANDLE;
GLOBAL_VARIABLES HWND DLLWINDOW CLHANDLE;
SET_FLCREATE DLLWINDOW FCFVAR=DLLWINDOW FCFFLAGS;
INITIALIZE DosQueryModuleHandle("DLLWINDOW FILENAME.dll",&DLLWINDOW MODULEHANDLE);
INITFUNCTION DosQueryModuleHandle("DLLWINDOW FILENAME.dll",&DLLWINDOW MODULEHANDLE);
REGISTER_CLASS WinRegisterClass(hab, "DLLWINDOW CLASS", (PFNWP) DLLWINDOW PROCEDURE, CS_SIZEREDRAW, 0);
INITFUNCTION WinRegisterClass(hab, "DLLWINDOW CLASS", (PFNWP) DLLWINDOW PROCEDURE, CS_SIZEREDRAW, 0);
DLLWINDOW HANDLE = WinCreateStdWindow(HWND_DESKTOP,
                                  WS_VISIBLE,
                                  &DLLWINDOW FCFVAR,
                                  "DLLWINDOW CLASS",
                                  "DLLWINDOW TITLE",
                                  null(),
                                  DLLWINDOW MODULEHANDLE,
                                  DLLWINDOW ID,
                                  (PHWND) & DLLWINDOW CLHANDLE);
                         /* Scale to Device: Set the window position */
Call WinSetWindowPos(DLLWINDOW HANDLE,
                HWND_BOTTOM,
                floor((DLLWINDOW XPOS * X_ScreenSZ) / X_DevelopSZ),
                floor((DLLWINDOW YPOS * Y_ScreenSZ) / Y_DevelopSZ),
                floor((DLLWINDOW XSIZE * X_ScreenSZ) / X_DevelopSZ),
                floor((DLLWINDOW YSIZE * Y_ScreenSZ) / Y_DevelopSZ),
                ior( SWP_MOVE , SWP_SIZE , SWP_ACTIVATE ) ) ;
END
```

Figure 113 (Part 3 of 3). PL/I Code Blocks for Device Independence

CODE/370 Compile and Link Procedures

We developed some additional REXX compile and link procedures for CODE/370.

Program PLICOMP

PLICOMP is a generalized compile procedure for PL/I programs. It invokes SQL and CICS preprocessors if requested.

The first compile option passed to the procedure directs the sequence of preprocessors:

```
pp(sql)             - invoke the SQL preprocessor
pp(sql cics)        - invoke the SQL and CICS preprocessors
pp(macro sql cics)  - invoke the PL/I macro processor, and SQL and CICS preprocessors
```

Example:

```
plicomp userid.CODE.PLI(progname) pp(sql cics) source test(all,sym)
```

```
/**** REXX. See The end of this file for comments  ********************/
trace 'o'

/* variables section       */
user = userid()
source  = ''                    /* will be set and change */
event   = 'SYSEVENT'            /* userid.SYSEVENT (not supported yet)*/
listing = 'PLISTNG'             /* userid.middle.PLISTNG(member)      */
object  = 'OBJ'                 /* userid.middle.OBJ(member)          */
macro   = 'MACRO'               /* userid.middle.MACRO.member (temp)  */
db2out  = 'DB2PBSRC'            /* userid.middle.DB2PBSRC    (member) */
db2list = 'DB2PCLST'            /* userid.middle.DB2LIST     (member) */
dbrmlib = 'DBRMLIB.DATA'        /* userid.DBRMLIB.DATA(member)        */
cicsout = 'CICSPLI'             /* userid.middle.CICSPLI(member)      */
cicslist= 'CICSLIST'            /* userid.middle.CICSLIST(member)     */

plipref = 'PLI370.DRIVER'       /* PL/I   prefix.SIELCOMP(IEL1AA)     */
lepref  = 'LE370.DRIVER'        /* LE/370 prefix.SCEExxx              */
cicspref= 'CICS330'             /* CICS   prefix                      */
db2pref = 'DSN230'              /* DB2    prefix                      */
inclib  = 'STL.CODE370.PLINCL'  /* include library                    */

arg sourcedsn options
if sourcedsn = ''  then do
   /* read command file from CODE/370 */
   parse source . . cmdname .
   cmdfile = user".EVFCMD."cmdname
   "ALLOC FI(CODECMDS) DA('"cmdfile"') OLD REUSE"
   "EXECIO 1 DISKR CODECMDS (FINIS STEM codcmd."
   rc1 = rc
   "FREE FI(CODECMDS)"
   if rc1 ¬= 0 then do
      say 'Failure in reading CODE/370 command parameter file:' rc1 cmdfile
      exit rc1;
   end
   parse upper var codcmd.1 plicmd sourcedsn options
end

sourcedsn = strip(sourcedsn,B,"'")
parse var sourcedsn pref1 '.' pref2 '(' member ')'
if member = ''  then do; say 'Member name not provided'; exit 16; end;
lastdot = lastpos('.',pref2)
pref3 = substr(pref2,lastdot+1)
pref2 = substr('.'pref2,1,lastdot)
```

Figure 114 (Part 1 of 4). Listing of Program PLICOMP.CMD

```
source     = pref1''pref2'.'pref3'('member')'
sourcelib  = pref1''pref2'.'pref3
objectdsn  = pref1''pref2'.'object'('member')'
macrodsn   = ''  /* pref1''pref2'.'macro'.'member */
listdsn    = pref1''pref2'.'listing'('member')'
db2outdsn  = pref1''pref2'.'db2out'('member')'
db2listdsn = pref1''pref2'.'db2list'('member')'
dbrmdsn    = pref1'.'dbrmlib'('member')'
cicsoutdsn = pref1''pref2'.'cicsout'('member')'
cicslistdsn= pref1''pref2'.'cicslist'('member')'
sysevent   = '"user"."event"'

options = strip(options,B)
if left(options,2) = 'PP' then do
   parse var options pp '(' processor ')' options
   options = strip(options,B)
   end
else processor = ''
processor = processor 'COMPILE'
options = options 'NOTERMINAL,SOURCE,INCLUDE'
db2opt  = 'HOST(PLI)'

  say 'Processors:         ' processor
  say 'Source:             ' source
  say 'Compile Options:    ' options

rc1 = 0
do i=1 to words(processor) until rc1 > 4
   pp = word(processor,i)
   say 'Invoking processor.............................' pp
   select
      when pp = 'MACRO'   then call PPMACRO
      when pp = 'SQL'     then call PPSQL
      when pp = 'CICS'    then call PPCICS
      when pp = 'COMPILE' then call COMPILE
      otherwise do
         say '===> invalid processor' pp '===> exit'
         exit 16
      end
   end /*select*/
end
if macrodsn ¬= '' then
   "DELETE '"macrodsn"'"
exit rc1

/*---------- DB2 preprocessor ---------------*/
PPSQL:
   say 'DB2  Input:         ' source
   say 'DB2  Trans Options: ' db2opt
   say 'DB2  Trans Source:  ' db2outdsn
   say 'DB2  Trans Listing: ' db2listdsn
   say 'DBRM Library:       ' dbrmdsn
   "ALLOC FI(SYSIN)    DSN('"source"')     SHR REUSE"
   "ALLOC FI(DBRMLIB)  DSN('"dbrmdsn"')    SHR REUSE"
   "ALLOC FI(SYSCIN)   DSN('"db2outdsn"')  OLD REUSE"
   "ALLOC FI(SYSPRINT) DSN('"db2listdsn"') OLD REUSE"
   "ALLOC FI(SYSLIB)   DSN('"sourcelib"' '"inclib"') SHR REUSE"
   "ALLOC FI(SYSUT1)   SPACE(1,1) CYLINDERS DELETE REUSE"
   "ALLOC FI(SYSUT2)   SPACE(1,1) CYLINDERS DELETE REUSE"
   "ALLOC FI(SYSTERM)  DUMMY REUSE"
   "CALL '"db2pref".DSNLOAD(DSNHPC)' " "'db2opt"'"
   rc1 = rc
   "FREE FI(SYSPRINT SYSCIN SYSIN SYSUT1 SYSUT2 DBRMLIB)"
   say 'DB2 translate return code:' rc1
   source = db2outdsn
   return

/*---------- CICS preprocessor ---------------*/
PPCICS:
   say 'CICS Input:         ' source
   say 'CICS Trans Source:  ' cicsoutdsn
```

Figure 114 (Part 2 of 4). Listing of Program PLICOMP.CMD

```
   say 'CICS Trans Listing: ' cicslistdsn
   "ALLOC FI(SYSIN)    DSN('"source"')      SHR REUSE"
   "ALLOC FI(SYSPRINT) DSN('"cicslistdsn"') OLD REUSE"
   "ALLOC FI(SYSPUNCH) DSN('"cicsoutdsn"')  OLD REUSE"
   "ALLOC FI(SYSTERM)  DUMMY REUSE"
   "CALL '"cicspref".SDFHLOAD(DFHEPP1$)' "
   rc1 = rc
   "FREE FI(SYSPRINT SYSPUNCH SYSIN)"
   say 'CICS translate return code:' rc1
   source = cicsoutdsn
   options = options 'SYSTEM(CICS)'
   return

/*---------- MACRO preprocessor --------------*/
PPMACRO:
   macrodsn   = pref1''pref2'.'macro'.'member
   say 'MACRO Input:        ' source
   say 'MACRO Output:       ' macrodsn
   "ALLOC FI(SYSIN)    DSN('"source"')    SHR REUSE"
   "ALLOC FI(SYSPRINT) DSN('"listdsn"')   OLD REUSE"
   "ALLOC FI(SYSLIB)   DSN('"sourcelib"' '"inclib"') SHR REUSE"
   "ALLOC FI(SYSPUNCH) DSN('"macrodsn"')  NEW CATALOG" ,
                       "RECFM(F,B) LRECL(80) BLKSIZE(6160)"
   "ALLOC FI(SYSUT1)   SPACE(1,1) CYLINDERS DELETE REUSE"
   "ALLOC FI(SYSTERM)  DUMMY REUSE"
   "CALL '"plipref".SIELCOMP(IEL1AA)'" ,
            "'"macro mdeck nosyntax noobject"'"
   rc1 = rc
   "FREE FI(SYSIN SYSLIB SYSPRINT SYSPUNCH SYSUT1)"
   say 'PLI macro return code:' rc1
   source = macrodsn
   return

/*---------- PLI compiler --------------------*/
COMPILE:
   say 'Compile Input:      ' source
   say 'Compile Listing:    ' listdsn
   say 'Object:             ' objectdsn
   "ALLOC FI(SYSIN)    DSN('"source"')    SHR REUSE"
   "ALLOC FI(SYSPRINT) DSN('"listdsn"')   OLD REUSE"
   "ALLOC FI(SYSLIN)   DSN('"objectdsn"') OLD REUSE"
   "ALLOC FI(SYSLIB)   DSN('"sourcelib"' '"inclib"') SHR REUSE"
   "ALLOC FI(SYSUT1)   SPACE(1,1) CYLINDERS DELETE REUSE"
   "ALLOC FI(SYSTERM)  DUMMY REUSE"
   /* if sysdsn(sysevent) = 'OK' then                          */
   /*   "ALLOC FI(SYSEVENT) DA("sysevent") OLD REUSE"          */
   /* else                                                     */
   /*   "ALLOC FI(SYSEVENT) DA("sysevent") NEW" ,              */
   /*          "RECFM(V) LRECL(4095) SPACE(10,10) TRACKS REUSE" */
   "CALL '"plipref".SIELCOMP(IEL1AA)'" "'"options"'"
   rc1 = rc
   "FREE FI(SYSIN SYSLIB SYSPRINT SYSLIN SYSUT1 SYSTERM)"
   say 'PLI compile return code:' rc1
   return


/*********************************************************************/
/* REXX:       PLICOMP                                               */
/*                                                                   */
/*             Licensed Materials - Property of IBM.                 */
/*                                                                   */
/*             5688-194 (C) Copyright IBM Corp. 1994.                */
/*             All rights reserved.                                  */
/*             US Government Users Restricted Rights - Use,          */
/*             duplication or disclosure restricted by GSA ADP       */
/*             Schedule Contract with IBM Corp.                      */
/*                                                                   */
/*             See Copyright Instructions.                           */
/*                                                                   */
/*                                                                   */
/* VERSION:    CODE/370  VERSION 1, RELEASE 2.0                      */
/*                                                                   */
```

Figure 114 (Part 3 of 4). Listing of Program PLICOMP.CMD

```
/* FUNCTION :  COMPILE a CICS DB2 PLI Program in Foreground          */
/*********************************************************************/
/* INPUT:                                                            */
/*                                                                   */
/* SOURCE:   PL/I source member of a dataset, for example            */
/*           userid.CODE.PLI(member)                                 */
/* OPTIONS:  PL/I  compiler options, specify through the Compile     */
/*           Options dialogs of CODE/370 Program Generator           */
/*           Includes invocation of preprocessors:                   */
/*             PP ( macro sql cics )                                 */
/*                                                                   */
/*********************************************************************/
/* INVOCATION: Through the CODE/370 Program Generator window         */
/*                                                                   */
/*********************************************************************/
/*  OUTPUT DATASETS: (previously allocated)           OPTION NEEDED  */
/*                                                      FOR OUTPUT   */
/*                                                                   */
/*  Object Deck:  userid.qual.OBJ(member)                            */
/*  Listing    :  userid.qual.PLISTNG(member)            SOURCE      */
/*  Preproc    :  userid.qual.DB2PCLST(member)           PP(SQL)     */
/*             :  userid.qual.CICSLIST(member)           PP(CICS)    */
/*                                                                   */
/*********************************************************************/
/* SUGGESTED NEXT STEP:                                              */
/*                                                                   */
/* Use LELINKF to Link/Edit object(s) into Load  module              */
/* Use BINDF   to bind DBRms into a plan                             */
/*********************************************************************/
```

Figure 114 (Part 4 of 4). Listing of Program PLICOMP.CMD

Program LELINKF

LELINKF is a generalized link procedure for PL/I programs with DB2 and/or CICS.

The first link-edit options passed to the procedure direct the inclusion of appropriate interface modules:

```
SQL        - link with DSNELI
CICS       - link with DFHELII
SQL CICS   - link with DFHELII and DSNCLI
```

Example:

```
lelinkf userid.CODE.OBJ(prog1,prog2) sql xref
```

```
/**** REXX. See The end of this file for comments  ******************/
trace 'o'

user = userid()
middle  = 'CODE'               /* userid.CODE.xxxx - default    */
load    = 'LOAD'               /* userid.middle.LOAD   (member) */
object  = 'OBJ'                /* userid.middle.OBJ    (member) */
lkedlist= 'LKEDLIST'           /* userid.middle.LKEDLIST(member)*/

lepref  = 'LE370.DRIVER'       /* lepref.SCEELKED, SCEECICS     */
db2pref = 'DSN230'             /* db2pref.DSNLOAD               */
cicspref= 'CICS330'            /* cicspref.SDFHLOAD             */
isppref = 'ISP.V3R5M0'         /* isppref.ISPLOAD               */
isrpref = 'ISR.V3R5M0'         /* isrpref.ISRLPA                */
```

Figure 115 (Part 1 of 3). Listing of Program LELINKF.CMD

```
arg sourcedsn linkopts
if sourcedsn = '' then do
                                  /* read command file from CODE/370 */
   parse source . . cmdname .
   cmdfile = user".EVFCMD."cmdname
   "ALLOC FI(CODECMDS) DA('"cmdfile"') OLD REUSE"
   "EXECIO 1 DISKR CODECMDS (FINIS STEM codcmd."
   rc1 = rc
   "FREE FI(CODECMDS)"
   if rc1 ¬= 0 then do;
      say 'Reading commands file failed:' rc1  cmdfile
      exit rc1;
   end
   parse upper var codcmd.1 lnkcmd sourcedsn linkopts
end

sourcedsn = strip(sourcedsn,B,"'")
parse var sourcedsn sourcedsn '(' member ')'
parse var sourcedsn pref1 '.' pref2
if member = '' then do; say 'Member name not provided'; exit 16; end;
if pref1  = '' then do; pref1 = user; pref2 = middle'.'object; end;

 linkopts = strip(linkopts,B)
 lecics = ''
 ledb2  = ''
 do i=1 to words(linkopts)
    if word(linkopts,i) = 'CICS' then do
       linkopts = delword(linkopts,i,1)
       lecics = ""lepref".SCEECICS'"
       leave
    end
 end
 do i=1 to words(linkopts)
    if word(linkopts,i) = 'SQL' then do
       linkopts = delword(linkopts,i,1)
       ledb2 = 'YES'
       leave
    end
 end
 if linkopts = '' then linkopts = 'NOTERM,MAP,LET'

 members = member
 members = translate(members,' ',',')
 member  = word(members,1)
 if lecics <> '' then members = members 'DFHELII'
 if ledb2  <> '' then if lecics <> '' then members = members 'DSNCLI'
                                      else members = members 'DSNELI'

 lastdot = lastpos('.',pref2)
 pref3 = substr(pref2,lastdot+1)
 pref2 = substr('.'pref2,1,lastdot)
 objectdsn = pref1''pref2'.'object
 loaddsn   = pref1''pref2'.'load
 listdsn   = pref1''pref2'.'lkedlist'('member')'

 if lecics <> '' then say 'Link with CICS Interface.....'
 if ledb2  <> '' then say 'Link with DB2  Interface.....'
 say 'Object:       ' objectdsn'('members')'
 say 'Load:         ' loaddsn'('member')'
 say 'LKED Listing: ' listdsn
 say 'Link Options: ' linkopts

 if words(members) = 1 then
       syslin = "'"objectdsn"("member")'"
 else do
      syslin = ''
      do i = 1 to words(members)
        syslin = syslin"'"objectdsn"("word(members,i)")' "
      end
 end
```

Figure 115 (Part 2 of 3). Listing of Program LELINKF.CMD

```
"ALLOC FI(SYSLIN)   DA("syslin") SHR REUSE"
"ALLOC FI(SYSLMOD)  DA('"loaddsn"("member")')   SHR REUSE"
"ALLOC FI(SYSTERM)  DUMMY REUSE"
"ALLOC FI(SYSPUNCH) DUMMY REUSE"
"ALLOC FI(SYSUT1)   SPACE(1,1) CYLINDERS DELETE REUSE"
"ALLOC FI(SYSPRINT) DA('"listdsn"') OLD REUSE"
"ALLOC FI(OBJ)      DA('"objectdsn"') SHR REUSE"
"ALLOC FI(SYSLIB)   DA('"loaddsn"'" ,
                      "'"db2pref".DSNLOAD'" ,
                      "'"isrpref".ISRLPA'"  "'"isppref".ISPLOAD'" ,
                      lecics ,
                      "'"lepref".SCEELKED'" ,
                      "'"cicspref".SDFHLOAD'" ,
                      "'SYS1.LINKLIB'             ) SHR REUSE"

"CALL 'SYS1.LINKLIB(IEWL)'" "'NOTERM,MAP,LET,"linkopts"'"
rc1 = rc

"FREE FI(SYSLIN SYSLMOD SYSTERM SYSPUNCH SYSUT1 SYSPRINT" ,
       "OBJ SYSLIB)"

 say 'Link-edit return code:' rc1

 if rc1 <= 4 then rc1 = '0'
 exit rc1;

/*********************************************************************/
/* REXX:       LELINKF                                               */
/*                                                                   */
/*             Licensed Materials - Property of IBM.                 */
/*                                                                   */
/*             5688-194 (C) Copyright IBM Corp. 1994.                */
/*             All rights reserved.                                  */
/*             US Government Users Restricted Rights - Use,          */
/*             duplication or disclosure restricted by GSA ADP       */
/*             Schedule Contract with IBM Corp.                      */
/*                                                                   */
/*             See Copyright Instructions.                           */
/*                                                                   */
/*                                                                   */
/* VERSION:    CODE/370  VERSION 1, RELEASE 2.0                      */
/*                                                                   */
/* FUNCTION :  Link/Edit object(s) in FOREGROUND                     */
/*                                                                   */
/*********************************************************************/
/* INPUT PARAMETERS:                                                 */
/*                                                                   */
/* SOURCE:   PL/I objects member of a dataset, for example           */
/*           userid.CODE.OBJ(memeber)                                */
/* LNKOPTS:  Link/Edit options specify through the User Options      */
/*           dialog of CODE/370 Program Generator                    */
/*                                                                   */
/*********************************************************************/
/* INVOCATION: Through the CODE/370 Program Generator window         */
/*                                                                   */
/*********************************************************************/
/*  OUTPUT DATASETS: (previously allocated)           OPTION NEEDED   */
/*                                                      FOR OUTPUT   */
/*                                                                   */
/*  Listing    :  userid.qual.LKEDLIST(member)           LIST        */
/*  Module     :  userid.qual.LOAD(member)                           */
/*                                                                   */
/*********************************************************************/
/* SUGESSED NEXT STEP:                                               */
/*                                                                   */
/* None                                                              */
/*                                                                   */
/*********************************************************************/
```

Figure 115 (Part 3 of 3). Listing of Program LELINKF.CMD

Program BINDF

BINDF is a procedure to perform the DB2 bind for multiple DBRMs into one plan.

Example:

```
bindf (prog1,prog2,prog3) planname REP CS
```

```
/**** REXX. See the end of this file for comments  ********************/
trace 'o'

/* variables section           */
user = userid()
db2sys   = 'DB23'              /* Default DB2 subsystem name          */
dbrmlib  = 'DBRMLIB.DATA'      /* Default DBRMLIB dataset             */

arg srcdsn plan action isolat
if srcdsn = '' then do
   /* Retrieve the bind request string (userid.EVFCMD.cmdname) */
   /* get the code command file */
   Parse source . . cmdname .
   cmdfile = user".EVFCMD."cmdname
   "ALLOC F(CODECMDS) DA('"cmdfile"') OLD"
   "EXECIO 1 DISKR CODECMDS (FINIS STEM codcmd."
   rc1 = rc
   "FREE F(CODECMDS)"
   if rc1 <> 0 then do
      say 'Reading commands file failed :' rc1  cmdfile
      exit rc1
   end
   parse upper var codcmd.1 bindf srcdsn plan action isolat
end
srcdsn = strip(srcdsn,B,"'")
parse var srcdsn srcdsn '(' members ')'
if members = '' then do                    /* member is required    */
   say 'No member specified'
   exit 16
end
members = translate(members,' ',',')
member  = word(members,1)
if srcdsn  = '' then srcdsn = user'.'dbrmlib
if plan    = '' then plan = member      /* default plan to 1st member */
if action  = '' then action = 'REP'     /* default is plan replace    */
if isolat  = '' then isolat = 'CS'      /* default is cursor stabil.  */

say 'DBRM Library:       ' srcdsn
say 'Members:            ' members
say 'DB2 subsystem name: ' db2sys
say 'Plan:               ' plan
say 'Action:             ' action
say 'Isolation:          ' isolat

/* Set up the db2 bind command */

"ALLOC FI(SYSTSPRT) DA('"user".TEMP1')",
    "RECFM(F,B) LRECL(133) SPACE(1,1) CYLINDERS" ,
    "BLKSIZE(3990) NEW DELETE"
"ALLOC FI(SYSPRINT) DA('"user".TEMP2')",
    "RECFM(F,B) LRECL(133) SPACE(1,1) CYLINDERS" ,
    "BLKSIZE(3990) NEW DELETE"
"ALLOC FI(SYSTERM) DUMMY REUSE"
"ALLOC FI(DBRMLIB) DA('"srcdsn"') SHR REUSE"
push " END"
push " BIND PLAN("plan") MEMBER("members") ACTION("action")" ,
          "ISOLATION("isolat")"
"DSN SYS("db2sys")"
rc1 = rc
"FREE FI(DBRMLIB,SYSTSPRT,SYSTERM,SYSPRINT)"
```

Figure 116 (Part 1 of 2). Listing of Program BINDF.CMD

```
say 'DB2 BIND return code is ' rc1
Exit rc1

/*********************************************************************/
/* REXX:       BINDF                                                 */
/*                                                                   */
/*             Licensed Materials - Property of IBM.                 */
/*                                                                   */
/*             5688-194 (C) Copyright IBM Corp. 1994.                */
/*             All rights reserved.                                  */
/*             US Government Users Restricted Rights - Use,          */
/*             duplication or disclosure restricted by GSA ADP       */
/*             Schedule Contract with IBM Corp.                      */
/*                                                                   */
/*             See Copyright Instructions.                           */
/*                                                                   */
/*                                                                   */
/* VERSION:    CODE/370  VERSION 1, RELEASE 2.0                      */
/*                                                                   */
/* FUNCTION :  BIND a DB2 PLAN in FOREGROUND                         */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/* INPUT PARAMETERS:                                                 */
/* The parameters required by this procedure are as follows          */
/*   cmdname = procedure name = BINDF                                */
/*   dataset = dbrm library dataset                                  */
/*   members = list of DBRMs that you want to bind into plan         */
/*   plan    = name of the plan that you are binding                 */
/*           = default is first member name                          */
/*   action  = ADD to add new plan or REP to replace existing plan   */
/*   isolat  = CS cursor stability or RR repeatable read             */
/*                                                                   */
/* The parameters are passed through the User Options dialog of the  */
/* Program Generator window                                          */
/*                                                                   */
/*********************************************************************/
/* INVOCATION:                                                       */
/*   CODE/370 Program Generator window                               */
/*                                                                   */
/*********************************************************************/
/*  OUTPUT DATASETS:                                                 */
/*                                                                   */
/*  DB2 system tables                                                */
/*                                                                   */
/*********************************************************************/
/*  SUGGESTED NEXT STEP:                                             */
/*                                                                   */
/*  After a successful BIND, the PLAN is ready for use. Two sample   */
/*  REXX procedures are available for DB2 program execution, and can */
/*  be found in the XXXXXX.SEQACLIS partition data set.              */
/*                                                                   */
/*  COBRUNDB : Debug a COBOL/DB2 program                             */
/*  CRUNDB   : Debug a C/DB2 program                                 */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/* MODIFICATION :                                                    */
/* 1. Modify the "variables section" according to your system        */
/*    environment                                                    */
/*                                                                   */
/*********************************************************************/
```

Figure 116 (Part 2 of 2). Listing of Program BINDF.CMD

Index

Special Characters

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

V

W

X

ITSO Technical Bulletin Evaluation RED000

International Technical Support Organization
PL/I for OS/2
PL/I for OS/2 Toolkit: Visual PL/I
CODE/370 PL/I Support
March 1995

Publication No. GG24-2501-00

Your feedback is very important to help us maintain the quality of ITSO Bulletins. **Please fill out this questionnaire and return it using one of the following methods:**

- Mail it to the address on the back (postage paid in U.S. only)
- Give it to an IBM marketing representative for mailing
- Fax it to: Your International Access Code + 1 914 432 8246

Please rate on a scale of 1 to 5 the subjects below.
(1 = very good, 2 = good, 3 = average, 4 = poor, 5 = very poor)

Overall Satisfaction ____

Organization of the book	____	Grammar/punctuation/spelling	____
Accuracy of the information	____	Ease of reading and understanding	____
Relevance of the information	____	Ease of finding information	____
Completeness of the information	____	Level of technical detail	____
Value of illustrations	____	Print quality	____

Please answer the following questions:

a) Are you an employee of IBM or its subsidiaries: Yes____ No____

b) Are you workin in the USA? Yes____ No____

c) Was the Bulletin published in time for your needs? Yes____ No____

d) Did this Bulletin meet your needs? Yes____ No____

If no, please explain:

_____

What other topics would you like to see in this Bulletin?

_____

What other Technical Bulletins would you like to see published?

_____

Comments/Suggestions: (THANK YOU FOR YOUR FEEDBACK!)

Name	Address
Company or Organization	
Phone No.	

ITSO Technical Bulletin Evaluation **RED000**
GG24-2501-00

Cut or Fold
Along Line

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

IBM International Technical Support Organization
Department 471, Building 070B
5600 COTTLE ROAD
SAN JOSE CA
USA 95193-0001

Fold and Tape **Please do not staple** Fold and Tape

Cut or Fold
Along Line

Printed in U.S.A.